Technische Universität Braunschweig
Institut für Software Systems Engineering

Diplomarbeit

Entwicklung einer Multi-Sensor-Datenfusion für ein autonomes Straßenfahrzeug

Sebastian Ohl
Matrikel-Nr. 2759270

Aufgabenstellung: Prof. Dr. B. Rumpe

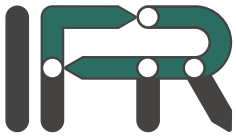
Betreuer: Dipl.-Ing. Jan Effertz
Dipl.-Wirt.-Inf. Christan Berger

Braunschweig, den 25. Juli 2007

Institut für Regelungstechnik

TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG

Prof. Dr.-Ing. W. Schumacher
Prof. Dr.-Ing. T. Form
Prof. em. Dr.-Ing. W. Leonhard



Diplomarbeit

Entwicklung einer Multi-Sensor-Datenfusion für ein autonomes Straßenfahrzeug

cand. inf. Sebastian Ohl

Betreuer	:	Dipl.-Ing. Jan Effertz
		Dipl.-Wirt.-Inf. Christan Berger
Eingereicht am	:	25. Juli 2007

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt zu haben. Die verwendete Literatur und sonstige Hilfsmittel sind vollständig angegeben.

Braunschweig, 25. Juli 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kurzfassung

In dieser Arbeit wurde ein System zur Wahrnehmung des Fahrzeugumfeldes im urbanen Gebiet entwickelt. Dieses wird im Rahmen des DARPA Urban Challenge von dem Team CarOLO eingesetzt. Die DARPA Urban Challenge ist ein Wettbewerb für autonome Fahrzeuge im Stadtverkehr. Die teilnehmenden PKWs müssen dabei Verkehrsregeln sowie andere Verkehrsteilnehmer berücksichtigen.

Das als Versuchsträger verwendete Fahrzeug, ein 2006 VW Passat, wurde dazu mit Laserscanner-, Radar- und Infrarotsensoren ausgerüstet. Diese Sensorkonfiguration bietet die Möglichkeit, ein vollständiges Bild des Fahrzeugumfelds zu erstellen. Durch Überschneidung der Sensorbereiche können die redundanten Messdaten genutzt werden, um eine wirksame Plausibilisierung der Sensormessdaten durchzuführen und so Falschdetektionen zu minimieren. Diese Eigenschaft ist von erheblicher Bedeutung, da im Vergleich zu Sensorkonzepten für die Autobahn sehr viel mehr ungewollte Objekte, beispielsweise durch Senken oder Hügel, von den Sensoren wahrgenommen werden.

Aktuelle Fahrerassistenzsysteme zielen eher auf außerstädtische Umgebungen, etwa im Rahmen des ACC, ab. Bei einer Ausweitung des Einsatzgebietes auf urbane Umgebungen ist eine weitaus komplexere Umfelderfassung vonnöten. Für ein Umfelderkennungssystem im urbanen Gebiet spielt, bedingt durch die vergleichsweise geringen Geschwindigkeiten, das nahe Umfeld um das Fahrzeug eine sehr bedeutende Rolle. Dieses muss neben dem Wissen über die Existenz von Hindernissen auch Informationen über die genauen Konturen der Objekte enthalten. Die in dieser Arbeit entwickelte Sensor-Daten-Fusion ermöglicht es, mehrere Sensorobjekte von verschiedenen Sensoren zu Konturen zusammenzufassen sowie die durch Laserscanner gewonnen Konturinformationen zu verwerten und mit Hilfe eines Extended Kalmanfilter zu tracken. Auf diese Weise kann das Fahrzeug sich in einem urbanen Umfeld mit höchster Genauigkeit bewegen.

Ein weiterer Schwerpunkt bei der Entwicklung dieses Systems ist das Softwaredesign. Die Grundlage der Architektur dieser Software stellt das Pipes and Filters Muster dar. Es stellt durch die strikte Trennung von Datenkanälen und Verarbeitungsstufen eine größtmögliche

Flexibilität zur Verfügung. Auf diese Weise ist es möglich, weitere Sensortypen durch Vorverarbeitungsstufen mit wenig Aufwand an die interne Datenrepräsentation anzupassen, ohne die anderen Verarbeitungsstufen zu verändern. Um den hohen Durchsatz von Sensorobjekten verarbeiten zu können ist ein intelligentes Speicherkonzept der verfolgten Objekte nötig. Zu diesem Zweck wurde eine objektorientierte Datenbank entwickelt, die den Zugriff von allen Stufen der Software optimal unterstützt.

Die hier entwickelte Sensor-Daten-Fusion stellt einen real funktionierenden Ansatz für die Umfelderkennung der nächsten Generation dar. Durch die Genauigkeit und modulare Struktur kann sie weiter entwickelt und in andere Systeme integriert werden.

Stichworte:

- Umfelderkennung
- Urbanes Umfeld
- DARPA Urban Challange
- Sensor-Daten-Fusion
- Extended Kalmanfilter
- Konturtracking

Aufgabenstellung

Die TU Braunschweig nimmt gegenwärtig an der DARPA Urban Challenge teil. Ziel dieses Wettbewerbes ist die autonome Fahrzeugführung im dynamischen Szenarien, wie z.B. dem Verkehrsfluss in einer Innenstadt. Für die Bewältigung dieser Aufgabe ist unter anderem eine konsistente Fahrzeugumfelderkennung vonnöten. Da kein Einzelsensor zum momentanen Zeitpunkt alle Anforderungen an eine solche Umfelderkennung erfüllen kann, werden verschiedene Sensoren mit gemeinsamem Beobachtungsbereich im Zuge einer Datenfusion miteinander kombiniert, um so die Stärken der einzelnen Sensorikkomponenten und deren Messprinzipien optimal ausnutzen zu können. Die in diesem Projekt eingesetzten Sensoren sind mit einer internen Objekterfassung (engl. Tracking) ausgestattet und liefern die in der Umwelt erkannten Merkmale als Messobjekte zurück.

Im Rahmen dieser Diplomarbeit sollen die Messdaten der verschiedenen Sensoren mit Hilfe eines Extended Kalmanfilters fusioniert werden, um ein zentrales Objekttracking zu realisieren. Teilaufgaben dieser Aufgabenstellung sind Datenzuordnung, Trackzustandsaktualisierung und Trackinitialisierung, die in geeigneten modularen Stufen in der Programmiersprache C++ umzusetzen sind. Die Stufen der Trackinitialisierung und Datenzuordnung sind dabei auf die gegebene Sensorkonfiguration anhand von Testfahrten zu optimieren. Eine geeignete Visualisierung des zentralen Trackings sowie der diesem zugrunde liegenden Messdaten schließt die Arbeit ab.

Über den Inhalt der Arbeit darf nur nach Rücksprache mit dem Institut für Regelungstechnik verfügt werden.

1 Einleitung

Ein Schwerpunkt heutiger Kraftfahrzeuge sind Fahrerassistenzsysteme. Gewöhnt sind Autofahrer inzwischen an Systeme mit beschränkten Eingriffen in das Fahrzeug wie beispielsweise das Antiblockiersystem oder die Antischlupfregelung. Diese Systeme haben die Sicherheit und den Fahrkomfort erheblich verbessert. Auch Systeme, die automatisches Einparken realisieren, sind am Markt verfügbar. In naher Zukunft werden Autos mit Spurhalteassistenten für Autobahnen ausrüstet sein. Für noch komplexeren Systeme ist eine gute Kenntnis des Fahrzeugumfeldes nötig. Dazu werden unterschiedliche Sensortypen eingesetzt, um ein konsistentes Bild der Welt zu erhalten. Die Zusammenführung all dieser Daten zu einem plausiblen Bild der Fahrzeugumgebung ist das Ziel dieser Arbeit.

1.1 DARPA Urban Challenge

Der Anlass zur Erstellung dieser Arbeit ist die Teilnahme der Technischen Universität Braunschweig an dem Wettbewerb DARPA Urban Challenge [Age06]. Dieser Wettbewerb ist der dritte Teil einer Reihe von Roboterwettbewerben. Der erste Teil der Serie fand 2004 unter dem Namen DARPA Grand Challenge statt. Ziel war es, dass ein autonomes Fahrzeug eine Strecke von 142 Meilen in einem Wüstengebiet zurücklegt. Diese Aufgabe hat keines der Fahrzeuge gelöst. Ein Jahr später waren fünf Fahrzeuge in der Lage, einen 132 Meilen langen Kurs durch ein Wüstengebiet ohne Eingriffe von Menschen zu bewältigen. Der aktuelle Wettbewerb besteht aus der Herausforderung, mit einem autonomen Fahrzeug innerhalb von sechs Stunden eine Strecke von 60 Meilen in einem urbanen Umfeld zurückzulegen. Dieser Wettkampf ist um vieles komplexer als die Aufgaben der ersten beiden Wettbewerbe. Er erfordert eine komplexere künstliche Intelligenz (KI), da beispielsweise Verkehrsregeln eingehalten werden müssen. Ferner werden sich alle autonomen Fahrzeuge gleichzeitig in dem Gelände bewegen, so dass mit Gegenverkehr und unerwarteten Reaktionen zu rechnen ist. Zuzüglich zu den rein verkehrstechnischen Aufgaben werden auch das Beherrschen von Einparksituationen oder das Navigieren auf einer Freifahrzone mit Hindernissen erwartet. Dazu sind sehr

viel komplexere Umfelderkennungssysteme nötig, als sie bei den ersten beiden Wettbewerben zum Einsatz kamen.

Die Motivation zur Durchführung dieser Wettbewerbe ist der National Defense Authorization Act der USA aus dem Jahr 2001, der das Ziel setzt, dass im Jahr 2015 ein Drittel der bodengestützen Kampffahrzeuge unbemannt und ferngesteuert sein soll. Ziel soll es sein, gefährliche Aufgaben von einer Maschine ausführen zu lassen.

1.2 Das CarOLO Projekt

Das CarOLO Projekt stellt den Rahmen für die Teilnahme der Technischen Universität Braunschweig an der DARPA Urban Challenge. An diesem Projekt sind fünf Institute der Universität mit jeweils unterschiedlichen Aufgabengebieten beteiligt. Das Institut für Software Systems Engineering übernimmt die Entwicklung der grundlegenden Frameworkarbeiten sowie die Überwachung des Softwareentwicklungsprozesses und das Projektmanagement. Für die Sensorik, Fahrzeugführung sowie das Sicherheitskonzept ist das Institut für Regelungstechnik verantwortlich. Die digitale Karte sowie die GPS-Navigation wird von dem Institut für Flugführung übernommen. Die an dem Versuchsträger installierten Kamerasysteme werden durch das Institut für Computergraphik ausgelesen und verarbeitet. Mit Hilfe dieser Daten wird unter anderem die Fahrspur erkannt. Das Institut für Betriebssysteme und Netze übernimmt die Aufgabe, die Rechnerinfrastuktur zu verwalten sowie die Entwicklung der künstlichen Intelligenz. Ein weiterer Partner in diesem Projekt ist die Ingenieursgesellschaft Auto und Verkehr mbH. Sie ist für Umbauarbeiten an dem Versuchsträger zuständig.

Insgesamt sind an diesem Projekt 35 Studenten und wissenschaftliche Mitarbeiter beteiligt. Sie müssen innerhalb eines Jahres die Anforderungen des Urban Challenge Wettbewerbs umgesetzen. Dies ist eine große Herausforderung, der das Team bis zum Site-Visit (Viertelfinale) gerecht wurde und sicher auch bis zum Finale im November 2007 noch gerecht werden wird.

1.3 Aufgabenbeschreibung

Um der Herausforderung gerecht zu werden, ist eine genaue Kenntnis des Fahrzeugumfeldes nötig. Ohne diese ist ein exaktes Navigieren im urbanen Gebiet nicht möglich. Diese Arbeit beschreibt eine Software zur Sensor-Daten-Fusion, die in dem Versuchsträger im Rahmen des Urban Challenge eingesetzt wird.

Ziel ist es, die eingehenden Sensordaten durch geeignete Filteralgorithmen und Regeln zu einem konsistenten Bild des Fahrzeugumfelds zusammenzufügen. Dazu wird der klassische Ansatz, bestehend aus Messwerterfassung, Zuordnung, Prädiktion, Filteraktualisierung und Trackinitialisierung, gewählt. Ein besonderer Schwerpunkt dieser Arbeit liegt in dem Umgang mit Multi-Sensor-Konfigurationen. Durch diese lassen sich Falschdetektionen von Objekten durch redundante Sensorabdeckungen erkennen und beheben oder die verfolgten Objekte zusätzlich absichern. Um genaue Navigationsinformationen für die künstliche Intelligenz erzeugen zu können, ist die hier entwickelte Software darauf ausgelegt, die Konturen der Objekte verarbeiten zu können. Dies ist ein erheblicher Fortschritt gegenüber früheren Arbeiten auf diesem Gebiet.

Ein weiterer Schwerpunkt dieser Arbeit ist im Feld der Softwarearchitektur zu finden. Die Gesamtstruktur des Software basiert auf dem Pipes and Filters Muster, das eine hohe Flexibilität und Skalierbarkeit bietet. Ferner wurde im Rahmen dieser Arbeit eine objektorientierte Datenbank zur optimalen Speicherung der verfolgenden Objekte entwickelt.

Die folgende Arbeit teilt sich in drei Teilbereiche auf. Zunächst werden in Abschnitt 2 die Grundlagen der Sensorik, Kalmanfilter sowie Tracking von Objekten vorgestellt. Daran schließt sich Abschnitt 3 über das Tracking von Fahrspuren an, das in dieser Arbeit am Rand behandelt wird. Abschnitt 4 geht detailliert auf die in dieser Arbeit entwickelten und verwendeten Algorithmen zum Tracking von Objekten sowie auf das Erstellen von Tracks ein. Die Architektur des CarOLO Gesamtprojekt sowie die genaue Architektur des in dieser Arbeit beschriebenen Teilprojekts, Sensor-Daten-Fusion, wird in Abschnitt 5 genau dargestellt.

2 Grundlagen

Dieses Grundlagenkapitel gibt eine Einführung in die in der weiteren Arbeit eingesetzten Verfahren. Dabei wird zunächst der verwendete Versuchsträger beschrieben. Er bildet die Grundlage für diese Arbeit. Darauf folgend werden die verwendeten Koordinatensysteme vorgestellt und anschließend auf ein Verfahren zur Objektverfolgung und Rauschreduktion eingegangen. Dieses wird in Abschnitt 2.4 vorgestellt und an Hand von Beispielen erklärt. In Abschnitt 2.5 wird dieses angewendet, um dynamische Objekte zu verfolgen.

2.1 Beschreibung der Hardware

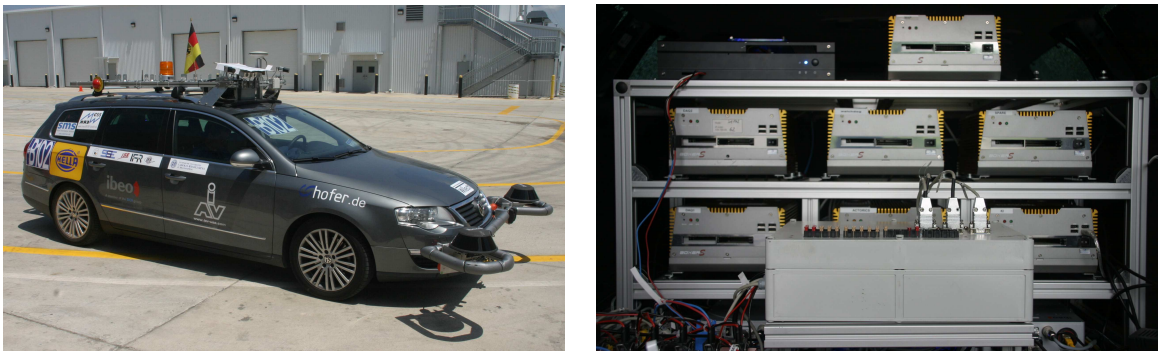


Abbildung 2.1: Versuchsträger, Kofferraum mit Rechnerrack

Der Versuchsträger ist ein 2006 VW Passat (Abbildung 2.1). Dieser wurde innerhalb des Projekts stark verändert, um ihn elektronisch ansteuern und mit den in Abschnitt 2.3 beschrieben Sensoren bestücken zu können. So wurden zum Teil Softwarekomponenten ausgetauscht, Datenbusse offengelegt und neue Stromversorgungen für 220 V und 24 V verlegt.

In dem Fahrzeug ist im Kofferraum ein Rack mit Rechnern integriert. Auf diesen laufen die in Abschnitt 5.1 beschrieben Programmteile. Die verbauten Rechner sind vom Typ Boxer S der Firma BRESSNER Technology GmbH. Diese eigenen sich für den Einsatz im Fahrzeug durch

ihre Temperaturbeständigkeit und Unempfindlichkeit gegenüber Erschütterungen. Dennoch bieten sie mit ihrem 2 Ghz Pentium M genug Rechenleistung, um alle Aufgaben innerhalb des Projekts ausführen zu können. Ferner ist ein Pentium Core 2 Duo als Rechner im Fahrzeug eingebaut.

Von den Rechnern des Typs Boxer S existieren sechs Stück im Fahrzeug. Ein Rechner enthält die Ansteuerung der Aktorik und die Rechnerüberwachung, einer die künstliche Intelligenz und die digitale Karte, zwei Rechner enthalten das Sensors Modul sowie die hier entwickelte Sensor-Daten-Fusion, zwei weitere Rechner sind derzeit noch ohne Aufgabe. Der Rechner, der den Core 2 Duo enthält, ist für die Verarbeitung von Bilddaten zur Fahrspurerkennung vorgesehen.

Verbunden sind diese Rechner über ein Gigabit-Ethernet-Netzwerk. Dieses bietet ausreichend Kapazität, um alle anfallenden Daten von ihren Quellen an die zugehörigen Datensenken zu verteilen. Als weitere Vernetzungsstruktur im Fahrzeug existieren diverse CAN Busse. Diese verbinden die Rechner mit der Sensorik oder den im Fahrzeug verlegten CAN Bussen zur Ansteuerung der Aktorik.

Ferner wurden unterschiedliche Sensorsysteme an das Fahrzeug angebracht. Ihre Anbaupositionen sowie Sichtbereiche sind in Abbildung 2.3 näher beschrieben. Am vorderen Stoßfänger sind zwei Laserscanner (IBEO Alaska XT) sowie ein Infratorsensor (Hella IDIS) befestigt. Ein Longrangeradar (SMS UMMR) ist am vorderen Teil des Dachgepäckträgers angebracht. Das hintere Fahrzeugumfeld wird von einem Laserscanner (IBEO LD ML) und einem weiteren Infrarotsensor beobachtet. Zusätzlich sind am hinteren Fahrzeug ein weiteres Longrangeradar sowie zwei Blindspotradarsensoren angebracht. Das hintere Longrangeradar befindet sich an dem hinteren Teil des Dachgepäckträges, während die Blindspotdetektoren über den Rückleuchten Platz finden.

2.2 Koordinatensysteme

In dieser Arbeit kommen drei unterschiedliche Koordinatensysteme zum Einsatz:

- innertiale Koordinaten
- innertiale kartesische Weltkoordinaten
- mitbewegte lokale kartesische Koordinaten

Das innertiale Koordinatensystem ist, basierend auf GPS-Daten, in WGS84 Koordinaten [oD04] beschrieben. Dieses bildet die Erde auf einem Ellipsoid ab (siehe Abbildung 2.2). Das

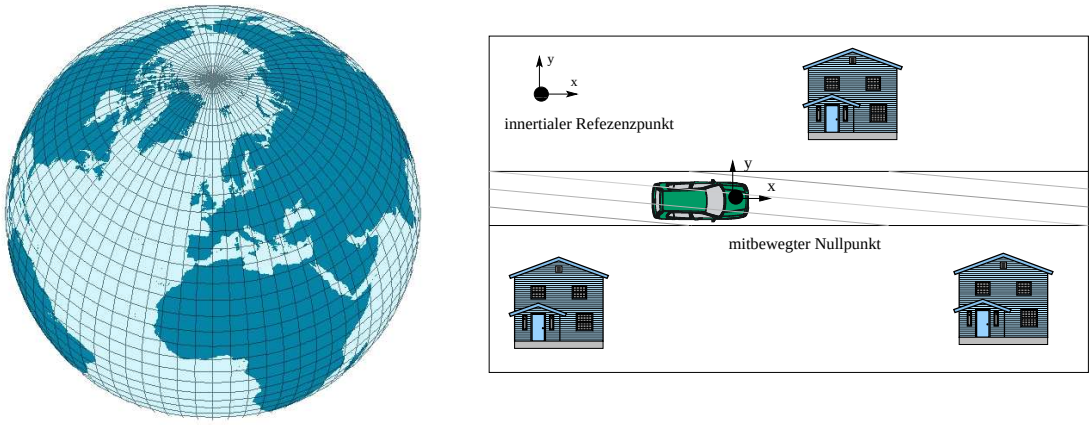


Abbildung 2.2: GPS Ellipsoid (rechts) [a], innertiale kartesische Weltkoordinaten und mitbewegte lokale kartesische Koordinaten (links)

[a]Stefan Kühn. Lizenziert unter Creative Commons CC-BY-SA 2.5.

Koordinatensystem wird dazu verwendet, die GPS-Position in kartesischen Weltkoordinaten zu bestimmen. Die GPS-Daten werden durch eine Trägheitsplattform gewonnen. Mit ihr ist eine Positionsbestimmung bis auf wenige Meter genau möglich. Hierzu werden verschiedene Sensoren, beispielsweise zur Beschleunigungsmessung, sowie ein Korrektursender genutzt.

Ein kartesisches ortsfestes Koordinatensystem (siehe Abbildung 2.2) vereinfacht die geometrischen Rechenoperationen erheblich. Hierbei bestehen nur lineare Beziehungen zwischen zwei Punkten. Aus diesem Grund beschreibt die in diesem Projekt entwickelte Software Objekte in kartesischen Weltkoordinaten. Da das GPS ein Koordinatensystem auf einem Ellipsoid beschreibt, ist eine Projektion nötig. Diese Projektion verankert das Koordinatensystem in einem festen, exakt eingemessenen innertialen Punkt. Mit steigenden Abstand zum Referenzpunkt steigt der Projektionsfehler. Dies kann für diesen Anwendungsfall allerdings vernachlässigt werden, da die Objekte der Umgebung im lokalen kartesischen Koordinatensystem gemessen werden und so lediglich einen Fehler gegenüber der Welt, nicht aber gegenüber dem Fahrzeug haben.

Das mitbewegte lokale kartesisches Koordinatensystem hat seinen Ursprung auf dem Mittelpunkt der Vorderachse des Fahrzeugs (siehe Abbildung 2.2). Alle Sensoren liefern ihre Objektdaten mit Bezug auf dieses System. Die Sensor-Daten-Fusion verarbeitet ihre Daten jedoch in innertialen kartesichen Koordinaten. So müssen diese entsprechend umgerechnet werden, um durch die Sensor-Daten-Fusion verarbeitet werden zu können. Die Entscheidung Objekte in innertialen und nicht in mitbewegten Koordinaten zu tracken, resultiert daraus, ortsfeste Objekte auch ortsfest zu beschreiben. Auf diese Weise werden nur bewegte Ob-

jekte mit Geschwindigkeitsvektoren beschrieben und nicht jedes stehende Objekt mit der Eigengeschwindigkeit versehen.

2.3 Sensorhardware

Das hier verwendetete Kraftfahrzeug besitzt drei Arten von Sensoren, deren Daten von der Sensor-Daten-Fusion verarbeitet werden: vier Radare des Firma Smart Microwave Sensors GmbH, zwei Infrarotsensoren der Firma Hella KGaA Hueck & Co sowie drei Laserscanner der Firma Ibeo Automobile Sensor GmbH. Jeder der verbauten Sensoren besitzt unterschiedliche Eigenschaften wie die Genauigkeit oder den Sichtbereich. In Abbildung 2.3 sind Anbau sowie Sichtbereiche markiert. Wie dort zu sehen ist, sind diese je nach Position und Art sehr unterschiedlich. Allen Sensorsysteme ist gemein, dass sie Objektdaten liefern. Das bedeutet, es wird bereits von den Steuergeräten eine Rohdatenauswertung und ein Tracking von Objekten durchgeführt. Im Folgenden werden die drei verwendeten Sensorsysteme kurz vorgestellt, um im Weiteren auf ihre Eigenschaften einzugehen.

2.3.1 SMS UMMR Radar

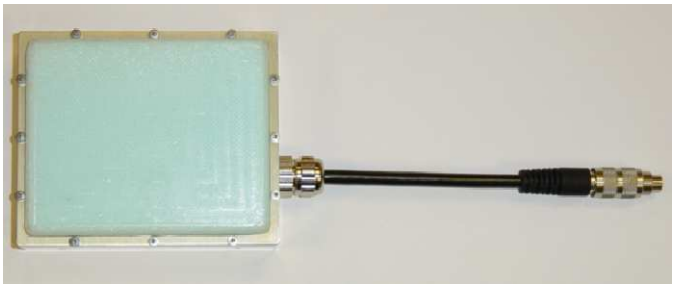


Abbildung 2.4: Das SMS UMMR Radar[a]

[a] Handbuch SMS UMMR Radar [sma06]

Das Universal Medium Range Radar (UMMR) (Abbildung 2.4) wurde entwickelt, um eine große Bandbreite an Anwendungen zu unterstützen. Besonderen Wert wurde dabei auf hoch dynamische Anwendungen gelegt.

Das UMMR kann in zwei unterschiedlichen Modi betrieben werden. Der Ultra Wide Band Pulse Mode ist für den Nahbereich gedacht. Dieser kann einen Bereich von 0.25m bis 15m beobachten und Geschwindigkeiten von $\pm 10 \frac{m}{s}$ messen. Dieser Modus ist vorwiegend für

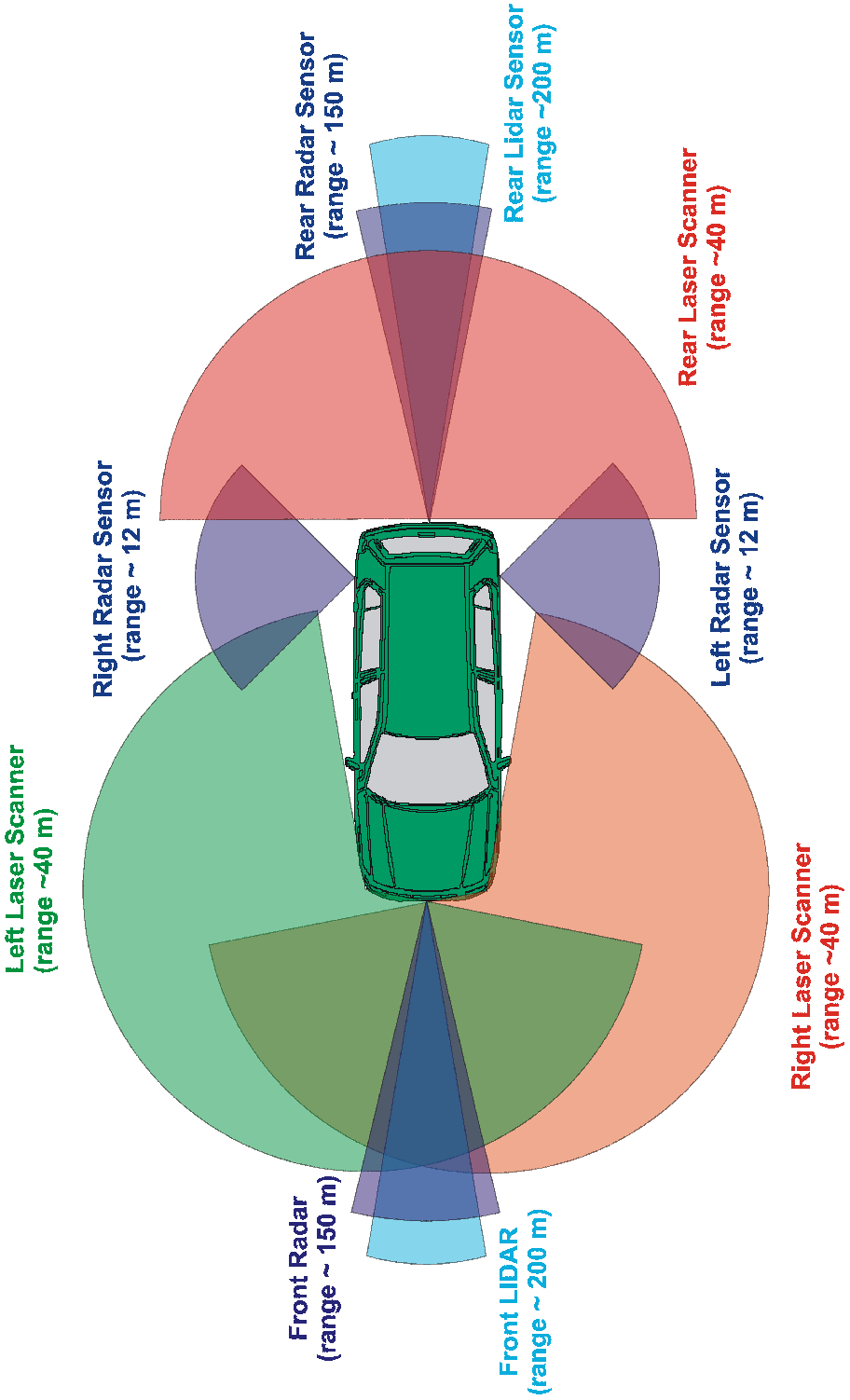


Abbildung 2.3: Von der Sensor-Daten-Fusion verarbeitete Sensorsysteme (nicht maßstabsgetreu)

Einparkanwendungen oder andere Nahbereichsbetrachungen gedacht. Von diesen Sensoren besitzt das Fahrzeug zwei Stück über den hinteren Rückleuchten.

Der zweite Modus des UMMR ist der Narrowband FMCW Mode. Dieser ist für den Fernbereich entwickelt worden und beobachtet einen Bereich zwischen 0.75m und mehr als 150m sowie Geschwindigkeiten im Bereich von $\pm 69.4 \frac{m}{s}$. Mit diesem Modus lassen sich z.B. Blind Spot Beobachtung oder Adaptive Cruise Control (ACC) Anwendungen realisieren. Das hier verwendete Fahrzeug besitzt zwei dieser Sensoren, die an der vorderen und hinteren Seite des Dachgepäckträgers angebracht sind.

Der Sensor kann maximal 32 Objekte verfolgen und wird über den CAN-Bus an die Data-Akquisitions-Schicht angebunden. Der Messfehler der Sensoren im FMCW Mode beträgt bei der Messung des Abstands 0.5m im Bereich von 0 bis 10m und ab 10m $\pm 5\%$. Die Geschwindigkeit wird mit einer Genauigkeit von $0.07 \frac{m}{s}$ und der Winkel mit einem Fehler von $< 0.5°$ angegeben [sma06]. Im UWB Mode beträgt der Fehler weniger als $\pm 0.135m$ des Abstands, $0.22 \frac{m}{s}$ der Geschwindigkeit sowie weniger als 2° Winkelfehler.

Für die hier entwickelte Applikation liefert das UMRR als Daten:

- X-Position in Fahrzeugkoordinaten
- Y-Position in Fahrzeugkoordinaten
- Geschwindigkeit in X-Richtung relativ zum Fahrzeug
- Geschwindigkeit in Y-Richtung relativ zum Fahrzeug

Die Radarsensoren bieten, neben dem grossen Sichtbereich, vor allem die Möglichkeit, Sensordaten anderer Sensoren zu verifizieren. Sie reagieren oft auf Objekte mit einem hohen Metallanteil. Beispielsweise werden durch Senkkastendeckel oft falsche Alarme ausgelöst. Mit Hilfe der anderen Sensoren lassen sich diese jedoch erkennen und nicht an die späteren Verarbeitungsstufen weiterreichen.

2.3.2 Hella IDIS Infrarot Sensor

Die in diesem Projekt verwendeten Sensoren der Firma Hella sind vom Typ IDIS (Abbildung 2.5). Sie werden normalerweise als ACC Sensor zur Abstandsmessung im Fahrzeug eingesetzt. Dort messen sie mit Hilfe von zwölf Laserpulsen den Abstand zu einem oder mehreren Hindernissen. Zu diesem Zweck wird eine Laufzeitmessung des Laserimpulses durchgeführt. Die zwölf Laser sind in einem Winkel von einem Grad gegenüber ihrem Nachbarn versetzt, so dass ein 12°-Öffnungswinkel durch den Sensor abgedeckt werden kann. Der Sensor ist in

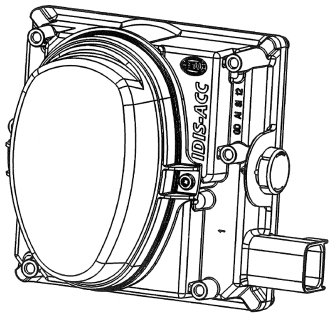


Abbildung 2.5: Hella IDIS Sensor[a]

[a]IDIS Handbuch [Hel06]

der Lage, maximal 25 Objekte mit bis zu 200m Abstand zu messen und dies mit einem Fehler von weniger als $\pm(1\% + 1m)$. Die laterale Auflösung beträgt 1°, entsprechend der Laserdioden. Der Sensor ist zusätzlich in der Lage, Geschwindigkeiten von Objekten durch Differentiation zu bestimmen.

Die durch den Sensor gelieferten Daten beschreiben den Abstand nach vorne sowie die linke und rechte Randposition des Objekts. Auf diese Weise werden linienförmige Objekte mit der Ausrichtung parallel zur Y-Achse des lokalen Koordinatensystems erzeugt.

Daraus ergeben sich für die Applikation folgende Objektdaten:

- X-Y-Position des rechten Randes in Fahrzeugkoordinaten
- X-Y-Position des linken Randes in Fahrzeugkoordinaten

Der IDIS Sensor hat sich in diesem Sensorkonzept für die Fernbeobachtung von Objekten bewährt. Durch ihn ist das Fahrzeug in der Lage, weiter als mit allen anderen Sensoren zu sehen. Die so gewonnen Daten werden im Nahbereich von den Radar- und Lasersensoren verbessert und verifiziert.

2.3.3 IBEO Laserscanner

Abbildung 2.6: IBEO Alaska XT (links), IBEO LD ML (rechts)

An dem Fahrzeug sind insgesamt drei Laserscanner der Firma IBEO Automobile Sensor GmbH verbaut. An dem vorderen Stossfänger sind zwei miteinander verschaltete Scanner des Type ALASCA XT (siehe Abbildung 2.6 (links)) angebracht. Diese bieten einen 260° Blick um das Fahrzeug und decken, wie in Abbildung 2.3 dargestellt, einen grossen Bereich im Nahfeld des Fahrzeugs ab. Am hinteren Stossfänger ist ein älteres Modell des Typs LD ML (siehe Abbildung 2.6 (rechts)) angebracht. Dieses deckt den hinteren Bereich, der den vorderen Scannern durch das Fahrzeug versperrt ist, ab.

Die Laserscanner messen mit Hilfe der Laufzeit des Lichts den Abstand zu reflektierenden Punkten. Auf diese Art lassen sich Punktwolken der Umgebung erstellen und zu den von den Sensoren gelieferten Konturen zusammensetzen. Die hier verwendeten Geräte sind Vier-Ebenen-Laserscanner. Sie messen in unterschiedlichen Winkeln mit dem Laser die Umgebung ab. Dies bietet der Objekterkennung die Möglichkeit, das „In-den-Boden-Schauen" der Sensorik zu erkennen (siehe Abbildung 2.7). Dazu werden nur Objekte weitergeleitet, bei denen Scanebenen ungefähr übereinander liegen. Die Laserscanner besitzen eine radiale Auflösung zwischen 0.125° (in Fahrtrichtung) und 1° (90° zur Fahrtrichtung).

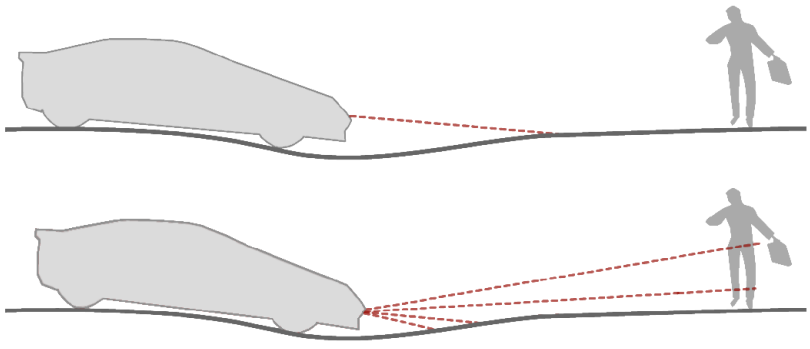


Abbildung 2.7: Unterschiede zwischen Ein- und Mehrebenen Laserscanner[a]

[a] Handbuch IBEO [Ibe06]

Im Gegensatz zu den Infrarot- und Radarsensoren liefern die Laserscanner nicht nur Punkte oder Linien sondern Objektkonturen. Diese beschreiben mit einer variablen Anzahl von Punkten die Außenkanten der erkannten Objekte. Ferner sind die Scanner in der Lage, über die Grösse und Form eines Objekts eine Klassifikation vorzunehmen sowie die Geschwindigkeiten zu messen.

Die Laserscanner liefern für die hier entwickelte Applikation folgende Daten:

- Eine Menge von X-Y-Punkten in Fahrzeugkoordinaten, die die Außenkante eines Objekts beschreiben
- Geschwindigkeit in X-Richtung relativ zum Fahrzeug
- Geschwindigkeit in Y-Richtung relativ zum Fahrzeug

Mit Hilfe der Laserscanner wird in diesem Sensorkonzept der Umwelt eine Kontur gegeben. Sie sind die einzigen Sensoren, die nicht nur Punkt- oder Linienobjekte erkennen können. Ohne sie wäre eine detaillierte Sicht auf das urbane Umfeld nicht möglich. Im Zusammenspiel mit den anderen Sensoren bieten sie jedoch die Herausforderung, die Daten zu kombinieren, da eine Wand von einem Laserscanner beispielsweise als ein Objekt, von einem Radar aber als mehrere Objekte wahrgenommen wird. Ein weiterer Vorteil ist, dass sie Objekte als zusammenhängend erkennen können. Bedingt durch Limitierungen in der internen Sensor-Vorverarbeitung besitzen sie gegenwärtig nur einen recht geringen Sichtbereich von ungefähr 40m Entfernung. Dies macht es schwer, sie zur Beobachtung des Gegenverkehrs oder bei hohen Geschwindigkeiten einzusetzen. Diese Schwäche wird jedoch durch die anderen am Fahrzeug verbauten Sensoren ausgeglichen.

2.4 Zustandsschätzung

Ein wichtiger Teil eines Umfelderkennungssystems ist das Tracking von Objekten mit Hilfe der durch Sensoren gewonnen Messdaten. Dazu wird ein Modell angenommen, das das Verhalten eines Objekts beschreibt. Damit ist eine Aussage über zukünftige Zustände des Objekts möglich, um so eine Zuordnung eines Messdatums zu einem bekannten Objekt zu erleichtern. Besagt das Modell beispielsweise, dass sich ein Objekt lediglich in eine Richtung bewegen kann, so kann die Bewegung des Objekts keine Kurve, sondern nur eine Gerade beschreiben. Mit Hilfe eines Modells lassen sich auch nicht messbare Zustandsgrößen bestimmen. So läßt sich über eine Positionsänderung eines Objekts und die dafür benötigte Zeit die gegenwärtige Geschwindigkeit bestimmen. Die Herausforderung bei der Erstellung eines

Modells ist, dass dieses die Realität möglichst gut abbilden muss und dennoch nicht zu kompliziert sein darf, um nicht einen zu grossen Zustandsraum zu besitzen. Letztes steigert den Rechenbedarf erheblich. Im Folgenden werden verschiedene Modelle und Filter vorgestellt, die das Verfolgen von Objekten ermöglichen.

Das folgende Beispiel beschreibt einen Zug, der durch seine zurückgelegte Strecke auf dem Gleis sowie seine gegenwärtige Geschwindigkeit beschrieben wird. Als Messwert wird die zurückgelegte Strecke zum Messzeitpunkt bestimmt. Dabei wird angenommen, dass dieser exakt der Realität entspricht, also kein Rauschen aufweist. Das Modell, das diesem Beispiel zugrunde liegt, ist, dass der Zug eine konstante Geschwindigkeit und deshalb keine Beschleunigung besitzt.

Auf diese Weise läßt sich seine Position folgendermaßen vorhersagen:

Zustandsvektor zum Zeitpunkt $k-1$:

$$\hat{x}_{k-1|k-1} = \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

Prädizierter Zustandsvektor zum Zeitpunkt k:

$$\hat{x}_{k|k-1} = \begin{pmatrix} x_{k-1|k-1} + \dot{x}_{k-1|k-1} \\ \dot{x}_{k-1|k-1} \end{pmatrix}$$

Zustandsaktualisierung auf den Zeitpunkt k:

$$\hat{x}_{k|k} = \begin{pmatrix} x_{mess} \\ x_{mess} - x_{k-1|k-1} \end{pmatrix}$$

Der Zustandsvektor $\hat{x}_{k|k}$ stellt den Zustand des Zuges nach einer Zeitscheibe dar. Wird eine neue Messung durchgeführt, wird der Messwert ohne Berücksichtigung der vorherigen Messungen übernommen und auf dessen Basis kann eine neue Schätzung erfolgen. Die in Abbildung 2.8 aufgetragenen Abweichungen zeigen, dass dieses Verfahren sehr gute Ergebnisse liefert, wenn der Zug sich entsprechend dem beschriebenen Modell verhält. Bei der Darstellung einer Zugfahrt mit sich ändernder Geschwindigkeit zeigt dieses Verfahren deutliche Abweichungen. Diese Abweichungen werden als Modellfehler bezeichnet.

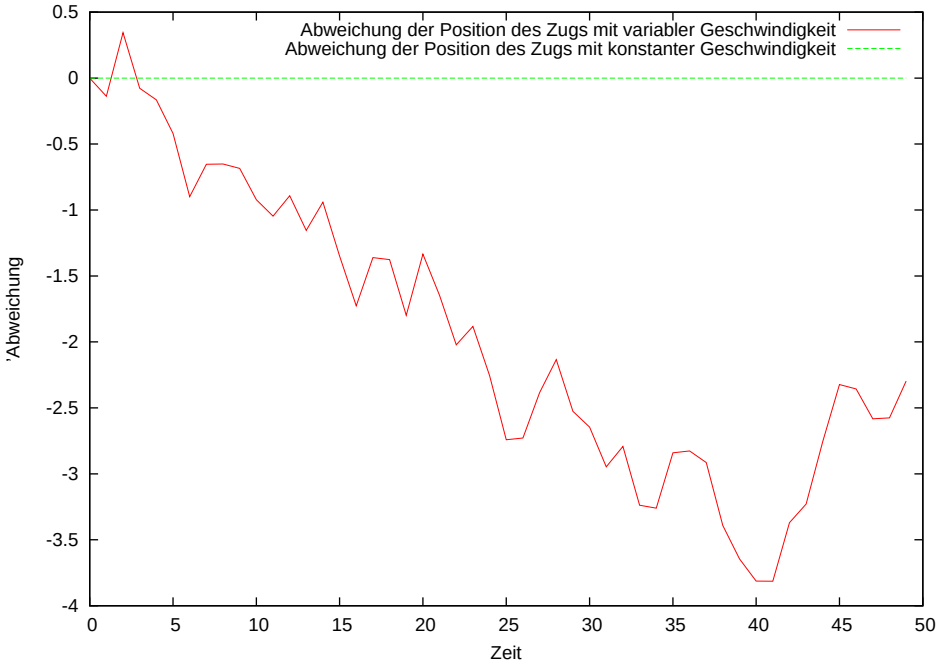


Abbildung 2.8: Abweichung der Zustandsschätzung von der realen Position eines Zugs ohne Filter mit und ohne variable Geschwindigkeit

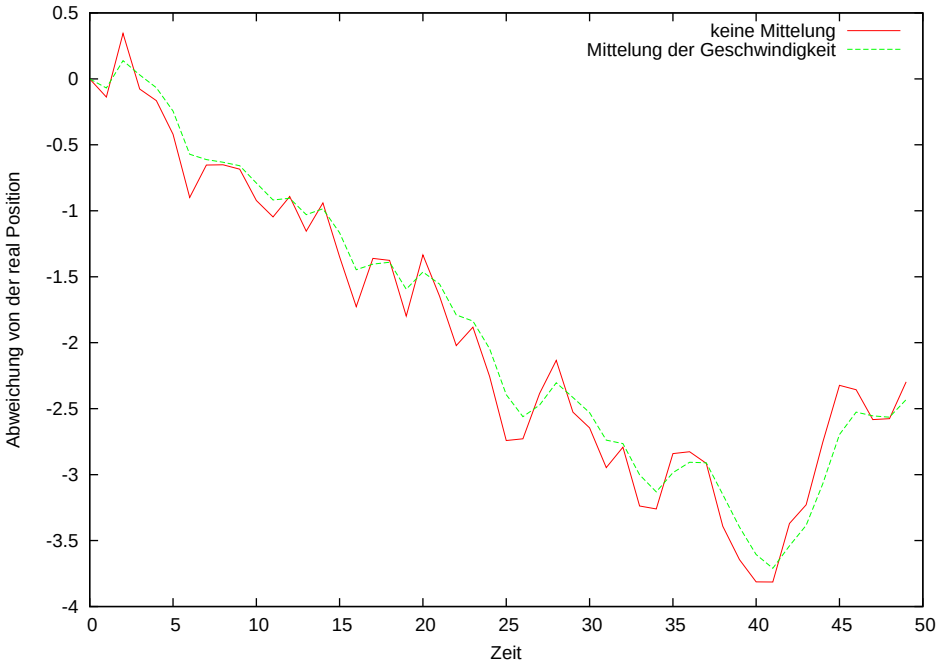


Abbildung 2.9: Abweichung der Zustandsschätzung von der realen Position mit gemittelter und nicht gemittelter Geschwindigkeit

Da rauschfreie Messungen in der Realität nicht möglich sind, werden Filtermechanismen benötigt, um dieses zu reduzieren und dennoch verlässliche Werte zu erhalten. Abbildung 2.9 zeigt die Abweichung der prädizierten Position gegenüber der Realposition bei gemittelter Geschwindigkeit. Dabei ist zu erkennen, dass Spitzen in den Abweichungen gegenüber den Abweichungen ohne Filter weniger stark ausgeprägt sind oder erst verzögert auftreten. Somit hat eine Rauschreduktion stattgefunden. Diese Verfahren wird Tiefpassfilter genannt. Die Zustandsaktualisierung nutzt hierbei die Geschwindigkeitsinformation des letzten Zeitschritts, um diese mit dem aktuellen Messwert zu kombinieren. Dabei wird das arithmetische Mittel beider Werte genutzt. Die Zustandsaktualisierung ändert sich damit zu:

Zustandsaktualisierung auf den Zeitpunkt k:

$$\hat{x}_{k|k} = \begin{pmatrix} x_{mess} \\ \frac{\dot{x}_{k-1|k-1} + (x_{mess} - x_{k-1|k-1})}{2} \end{pmatrix}$$

Die in dieser Arbeit eingesetzte Möglichkeit, dem Rauschen der Messwerte zu begegnen, ist der Kalmanfilter [Kal60]. Dieser ist ein Verfahren, das mit Hilfe eines linearen Systemmodells und der Unsicherheiten der Messungen und des Modells eine im statistischen Sinne optimale Schätzung des Zustandsvektors ermöglicht. Dazu wird die Unsicherheit in Kovarianzmatrizen abgelegt, der ein normalverteiltes, mittelwertfreies und unkorreliertes Rauschen zugrunde liegt.

Das den vorherigen Beispielen zugrunde liegende Bewegungsmodell des Zugs läßt sich auch für den Kalmanfilter nutzen. Hierzu ist neben dem bereits bekannten Zustandsvektor noch eine Systemmatrix zu definieren. Sie beschreibt das Bewegungsmodell und wird genutzt, um von einem auf den nächsten Zeitschritt zu prädizieren.

Für dieses Bewegungsmodell lautet die Systemmatrix:

$$F_k = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Zusätzlich wird eine Beschreibung der Unsicherheiten des Modells benötigt. Dieses wird durch eine Kovarianzmatrix gelöst. Sie ergibt sich durch die Nutzung des Modells bzw. dadurch, dass das verfolgte Objekt sich nicht entsprechend dem Modell verhält. Wie in den vorherigen Bespielen gezeigt, beschreibt das verwendete Systemmodell eine Zugfahrt mit konstanter Geschwindigkeit, der Zug bewegt sich jedoch mit konstanter Beschleunigung. Dies läßt sich durch eine höhere Varianz ausdrücken.

Das Beispiel nutzt folgende Kovarianzmatrix:

$$Q = \begin{pmatrix} 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}$$

Der Prädiktionsschritt des Kalmanfilters teilt sich in zwei Formeln auf. Die eine prädiziert den Zustandsvektor, die andere prädiziert die Kovarianzmatrix.

Prädiktion des Zustandsvektors:

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1}$$

Prädiktion der Kovarianzmatrix:

$$P_{k|k-1} = F_k * P_{k-1|k-1} * F_k^T + Q_k$$

Der Schritt der Zustandsaktualisierung ist sehr viel komplexer als bei den bisherigen Beispielen. Er muss auch die Aktualisierung der Varianzmatrizen durchführen. Wie aus der Formel zur Prädiktion der Kovarianz zu sehen ist, erhöht sich die Varianz mit der Anzahl der durchgeführten Prädiktionen. Aus diesem Grund ist es nötig, die prädizierten Werte so oft wie möglich mit gemessenen Werten abzugleichen. Die Zustandsaktualisierung läßt sich in zwei Fälle teilen.

Fall 1: Kein Messwert vorhanden

Im Fall, dass kein Messwert vorhanden ist, wird der Zustandsvektor und die Kovarianzmatrix des Kalmanfilters lediglich mit den prädizierten Werten aktualisiert.

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} P_{k|k} = P_{k|k-1}$$

Hierbei vergrößert sich die Varianz des Zustands, da bei der Prädiktion der Varianz die Modellvarianzen hinzukommen. Dieser Fall sollte nicht zu häufig vorkommen, da die Werte sonst zu unsicher werden.

Fall 2: Messwert vorhanden

In diesem Fall wird die Innovation ($\tilde{y}_k$) des Kalmanfilters mit Hilfe des neuen Messwerts (z_k) bestimmt. Hierzu muss zunächst das Messdatum aus dem Messraum in den Zustandsraum

projiziert werden. Dies geschieht mit Hilfe der Messmatrix H_k. In diesem Beispiel wird die zurückgelegte Strecke des Zugs gemessen. Dafür lautet die Messmatrix:

$$H_k = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

Die Innovation berechnet sich damit zu:

$$\tilde{y}_k = z_k - H_k * \hat{x}_{k|k-1}$$

Zur Bestimmung der Kovarianz der Innovation (S_k) wird ferner das Rauschen des Messwerts (R_k) benötigt. Die Kovarianz der Innovation ergibt sich damit zu:

$$S_k = H_k * P_{k|k-1} * H_k^T + R_k$$

Das Kalman Gain (K_k) beschreibt eine Verstärkungsmatrix, den Einfluss des Messvektors und der Prädiktion auf den aktualisierten Zustandsvektor und die Kovarianz. Für das Kalman Gain wird nur die Kovarianz der Messung und des Filters benötigt, nicht aber der Messwert. Dies leitet sich daraus ab, dass es nur eine Gewichtung zwischen Modell und Realität angibt. Diese leitet sich somit direkt aus der Unsicherheit des Modells, das in der Kovarianz des Filter abgebildet wird, ab.

$$K_k = P_{k|k-1} * H_k^T S_k^{-1}$$

Die eigentliche Aktualisierung der Varianzen und des Zustandsvektors ergibt sich zu:

aktualisierter Zustandsvektor:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k * \tilde{y}_k$$

aktualisierte Kovarinz:

$$P_{k|k} = (I - K_k * H_k) * P_{k|k-1}$$

Abbildung 2.10 zeigt die Abweichung von der Realposition des Zugs, der durch einen Kalmanfilter getrackt wurde. Es ist eine deutlich geringere Abweichung festzustellen als bei einem Tracking mit den bisher beschriebenen Algorithmen.

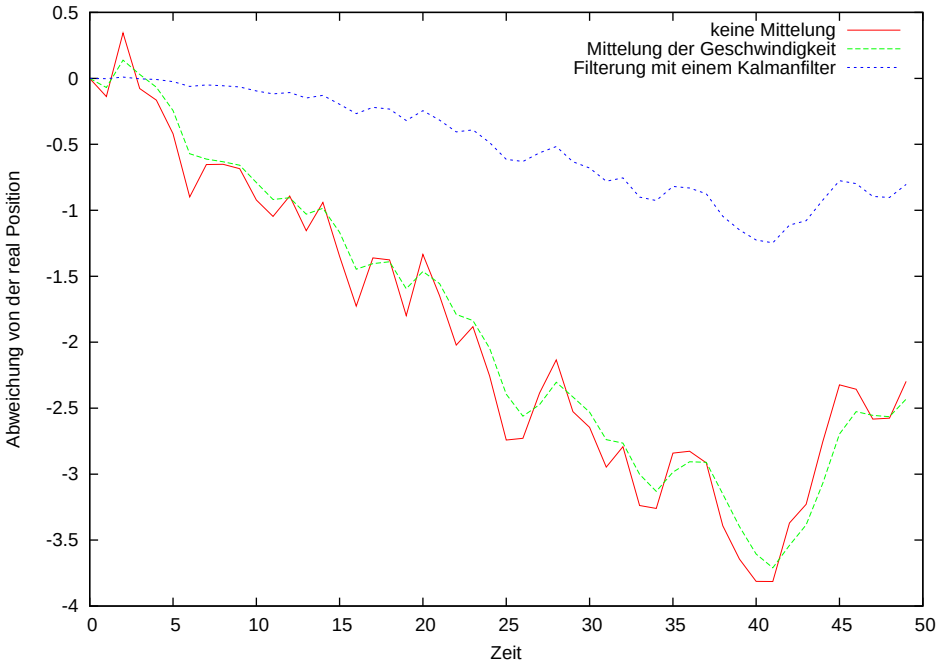


Abbildung 2.10: Abweichung der Zustandsschätzung von der realen Position mit gemittelter und nicht gemittelter Geschwindigkeit sowie Kalman-gefilterter Geschwindigkeit

Wie bereits beschrieben, benötigt der Kalmanfilter ein lineares Systemmodell. Da in dieser Arbeit aber nicht nur lineare, sondern auch nicht lineare Modelle zum Einsatz kommen, wird zusätzlich zum normalen Kalmanfilter auch der erweiterte Kalmanfilter [Bal99] eingesetzt. Er ist in der Lage, diese Modelle zu verarbeiten. Dabei kann das Modell in zwei unterschiedliche Modelle aufgespalten werden: das Beobachtungs- und das Zustandsmodell. Jedes kann für sich linear oder nicht linear sein. Im nicht linearen Fall wird eine Linearisierung beispielsweise mit einer Taylorreihenentwicklung nötig.

Die Linearisierungsfunktionen $f(x_{k-1})$ und $h(x_k)$ können nicht direkt genutzt werden, um die Kovarianz zu aktualisieren. Stattdessen wird in jedem Schritt für F_k und H_k eine Jacobimatrix berechnet. Diese kann in den Kalmanfiltergleichungen genutzt werden.

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1|k-1}} \quad H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k|k-1}}$$

Da sich der erweiterte Kalmanfilter durch das Einlinienmodell nicht leicht auf das Zugbeispiel anweden lässt, wird hier ein anderes Systemmodell genutzt. Es beschreibt einen Punkt im Raum, beispielsweise ein Fahrzeug, das sich mit konstanter Geschwindigkeit (v) in eine konstante Richtung (α) bewegt. Der Geschwindigkeitsvektor wird in Polarkoordinaten

angegeben. Somit ergibt sich der Zustandsvektor zu:

$$\hat{x}_k = \begin{pmatrix} x \\ y \\ v \\ \alpha \end{pmatrix}$$

Da v und α nicht linare Zusammenhänge zu den übrigen Zustandsgrößen haben wird der erweiterte Kalmanfilter nötig. Hierbei muss die Systemmatrix linearisiert werden. Dies geschieht mit einer Taylorreihe.

Die Systemmatrix bestimmt sich damit zu:

$$F_k = \begin{pmatrix} 1 & 0 & \cos(\alpha) & -v * \sin(\alpha) \\ 0 & 1 & \sin(\alpha) & v * \cos(\alpha) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2.5 Tracking von Objekten

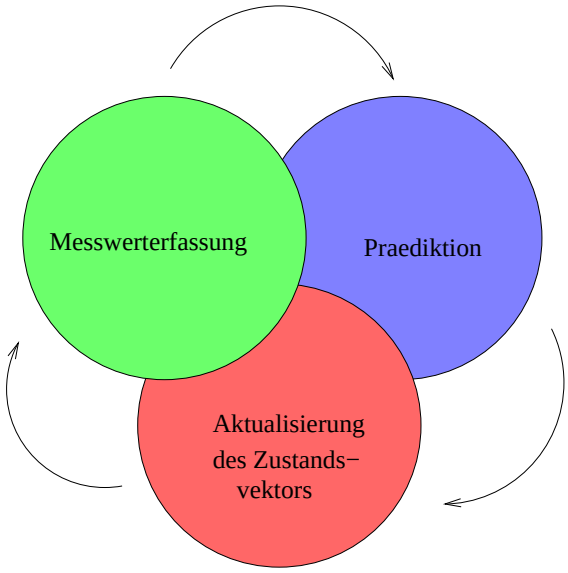


Abbildung 2.11: Tracking als immer wiederkehrende Aufgabe

Die Verfolgung von Objekten bezeichnet man als Tracking. Im Allgemeinen wird jedem Objekt ein Menge von Parametern zugewiesen, die es zu verfolgen gilt (z.B. die Position oder

Geschwindigkeit). Das Tracking von Objekten teilt sich in drei Schritte auf. Diese werden immer wieder ausgeführt. Abbildung 2.11 beschreibt zunächst die Bestimmung eines Messwerts durch einen Sensor. Darauf folgt die Prädiktion der Messdaten aus dem dem Objekt zugewiesenen Zustandsvektor, um beispielsweise Plausibilätsüberprüfungen durchführen zu können. Der dritte und letzte Schritt aktualisiert mit Hilfe des Messwerts und der Prädiktion den Zustandsvektor (z.B. mit einem Kalmanfilter). Diese drei Schritte werden immer in dieser Reihenfolge durchgeführt, bis sich das Objekt aus dem Beobachtungsbereich entfernt hat.

Im Folgenden wird auf das Verfolgen von Objekten eingegangen. Im Rahmen dieses Kapitels werden die Grundlagen vorgestellt. Die genauen Algorithmen zur Verfolgung von Objekten werden Abschnitt 4 behandelt.

Ein Objekt kann gegenüber dem Weltkoordinatensystem eine Eigenbewegung aufweisen. Wird ein Objekt als zu einem Zeitpunkt x an einem Ort gemessen, kann es sich potenziell zum Zeitpunkt $x + 1$ an einem anderen Ort befinden. Die Aufgabe des Trackings ist es, die Bewegung eines Objekts zu beschreiben und dieses bei einer späteren Messung erneut zu identifizieren.

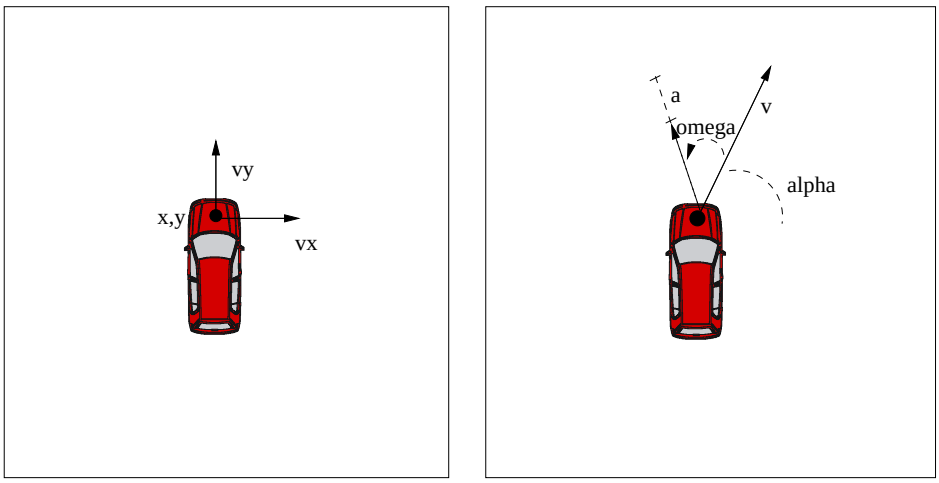


Abbildung 2.12: Eingesetzte Bewegungsmodelle, links Pretrackingmodell, rechts Haupttrackingmodell

In der Literatur [Bro98] wird ein Objekt als Punkt im Raum beschrieben. Dieser Punkt hat in der Regel eine Position sowie Eigengeschwindigkeiten gegenüber dem Weltkoordinatensystem. Das in diesem Projekt beschriebene nicht lineare Bewegungsmodell (siehe Abbildung 2.12 (rechts)) besitzt sechs Dimensionen: Die Position in innertialen 2D-Weltkoordinaten,

die Geschwindigkeit sowie die Beschleunigung und den Kurswinkel sowie die Winkeländerungsgeschwindigkeit. Dieses wird in einem erweiterten Kalmanfilter ausgedrückt. Während des Pretrackings wird ein einfacheres Modell verwendet. Dieses beschreibt ein Objekt mit einem linearen Bewegungsmodell. Es hat vier Dimensionen: Die Position in innertialen 2D-Weltkoordinaten, die Geschwindigkeit in X-Richtung und die Geschwindigkeit in Y-Richtung. Es wird in Abschnitt 5.3.5 näher beschrieben.

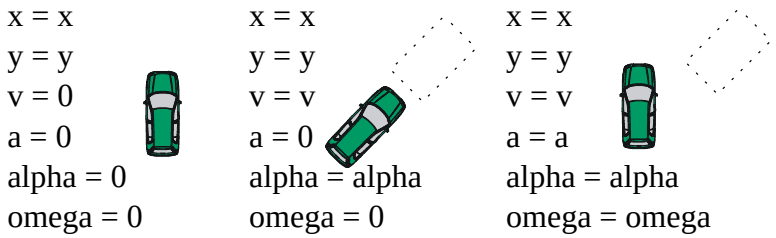


Abbildung 2.13: Mögliche Bewegungen eines Objekts, keine Bewegung, lineare Bewegung, nicht lineare Bewegung (von links nach rechts)

Das Systemmodell des Kalmanfilters ergibt sich damit zu:

$$\hat{x}_k = \begin{pmatrix} x \\ y \\ v \\ \dot{v} = a \\ \alpha \\ \dot{\alpha} = \omega \end{pmatrix}$$

$$P_k = \begin{pmatrix} 1 & 0 & \cos(\alpha) * \Delta T & 0 & -v * \sin(\alpha) * \Delta T & 0 \\ 0 & 1 & \sin(\alpha) * \Delta T & 0 & v * \cos(\alpha) * \Delta T & 0 \\ 0 & 0 & 1 & \Delta T & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta T \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Dieses Modell ist in der Lage, ein punktförmiges Objekt zu beschreiben, das sich im Vergleich zu dem Zug aus Abschnitt 2.4 sowohl in X- als auch in Y-Richtung bewegen kann. Darüber hinaus bildet es auch Geschwindigkeitsänderungen oder eine Richtungsänderung des Objekts ab. Bezogen auf die Aufgabe des Verfolgens von Objekten im Straßenverkehr, bildet es alle normalen Bewegungen von anderen Verkehrsteilnehmern ab. So kann das Modell ein Objekt verfolgen, das sich auf einer Straße linear vorwärts bewegt. Dabei ist es auch in der

Lage, das Bremsen oder Beschleunigen eines Objekts zu prädizieren. Über den Parameter ω lässt sich eine Kurvenfahrt eines Objekts, z.B. beim Abbiegen oder Spurwechsel, abbilden. Abbildung 2.13 zeigt die möglichen Bewegungen in Zusammenhang mit den möglichen Zustandsgrössen.

Wie im Abschnitt 2.3 beschrieben, liefern die an dem Versuchsfahrzeug eingesetzten Sensoren nicht ausschließlich punktförmige Objekte, sondern auch Linien und Konturen eines Objekts. Dieses lässt sich nicht mit dem beschrieben Filter abbilden. Hierfür wird der Zustandsvektor um ein X-Y-Paar für jeden Punkt einer Kontur erweitert und dann davon ausgegangen, dass sich eine Kontur nur als Ganzes bewegt. Dieses wird in Abschnitt 4.3 näher beschrieben.

3 Tracken von Fahrspuren

Im Rahmen dieser Arbeit wurde als Beispiel für Tracking von ortsfesten Objekten eine Fahrspurverfolgung realisiert. Diese wurde in der Fahrspurerkennung des Versuchsträgers eingesetzt. Fahrspuren haben, im Vergleich zu dynamischen Objekten, eine feste Position in einem Koordinatensystem. In diesem Projekt werden sie in dem hier verwendeten kartesischen Weltkoordinatensystem beschrieben. Da sie ortsfest sind, besteht ihr Zustandsvektor nur aus den jeweiligen Koordinaten. Eine Beschreibung einer Eigendynamik ist nicht nötig.

Zustandsvektor von ortsfesten Objekten:

$$\hat{x}_k = \begin{pmatrix} x \\ y \end{pmatrix}$$

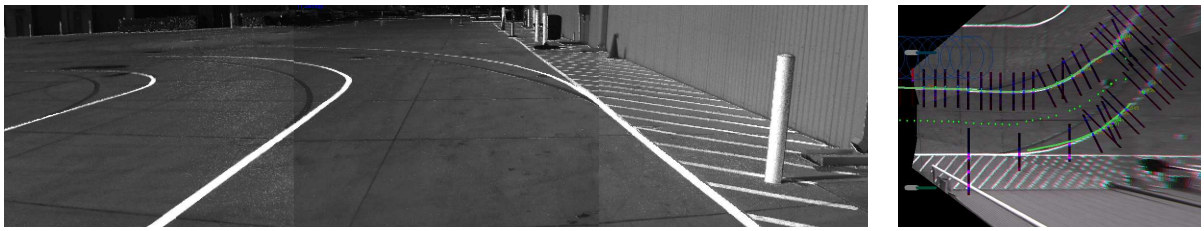


Abbildung 3.1: Tracking zweier Fahrspuren, Orginalbild (links), transformiertes Bild (rechts)

Bei dieser Applikation bekommt die Trackingstufe ein bereits aufgearbeitetes Bild, in dem sie Messungen durchführen kann. Dieses Bild ist bereits von dem Kamerabild entsprechend einer Kalibrierung in eine Ebene mit mitbewegten lokalen Koordinatensystem projiziert und durch verschiedene Filter von Hintergrund und Störungen gereinigt. Abbildung 3.1 stellt ein solches Bild dar. Das linke Bild stellt das unbearbeitete Kamerabild dar, das rechte Bild zeigt das verarbeitete Bild. Hier sind die durchgeführten Messungen als schwarze Balken zu erkennen. Die erkannte Fahrspur ist durch grüne Splines dargestellt.

Eine Fahrspur wird durch einen BSpline beschrieben, der durch getrackte Stützstellen beschrieben wird. Jede Fahrspur (links und rechts) besitzt somit folgendes Systemmodell:

$$\hat{x}_k = \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_n \\ y_n \end{pmatrix}$$

$$P_k = \begin{pmatrix} \sigma_{x1} & \sigma_{x1y1} & 0\dots & & & & \\ \sigma_{y1x1} & \sigma_{y1} & 0\dots & & & & \\ & \dots 0 & \sigma_{x2} & \sigma_{x2y2} & 0\dots & & \\ & \dots 0 & \sigma_{y2x2} & \sigma_{y2} & 0\dots & & \\ & \dots 0 & 0 & 0 & \ddots & & \\ & & & & \dots 0 & \sigma_{xn} & \sigma_{xnyn} \\ & & & & \dots 0 & \sigma_{ynxn} & \sigma_{yn} \end{pmatrix}$$

Abbildung 3.2 beschreibt den Vorgang, der bei Eintreffen eines neuen Bildes durchgeführt wird. Zunächst wird eine Aktualisierung der Stützpunkte durchgeführt. Dabei werden zuerst Punkte, die hinter dem Fahrzeug liegen, entfernt, da das Fahrzeug diese bereits passiert hat.

Existieren anschließend keine Punkte mehr in dem Spline, so wird entsprechend der erwarteten Position (linke oder rechte Seite des Fahrzeugs) nach neuen Fahrspuren gesucht. Wie in Listing 3.1 beschrieben, werden drei mögliche Verläufe einer Fahrspur (linke/rechte Kurve und eine Gerade) gemessen und der Verlauf, der die meisten gültigen Messpunkte ergibt, als neue Fahrspur angenommen.

Nun wird für jede Stützstelle des Splines ein Messwert bestimmt. Dies entspricht dem ersten Schritt aus Abbildung 2.11. Liegt die Qualität des Messwerts über einer Schwelle, so wird Schritt zwei und drei ausgeführt und der getrackte Stützpunkt mit Hilfe eines Kalmanfilters aktualisiert. Liegt die Qualität darunter, so wird ein Zähler erhöht, der angibt, wie oft ein Messpunkt nicht gefunden wurde. Ab einem Schwellwert wird der Punkt entfernt.

Der nächste Schritt ist das Finden von weiteren Punkten zum Stützen der Fahrspur. Zu diesem Zweck wird eine bestimmte maximale Länge (ca. 20m) der Fahrspur angestrebt. Diese Länge definiert sich durch die Sichtweite der Kamera. Der Spline wird linear interpoliert und

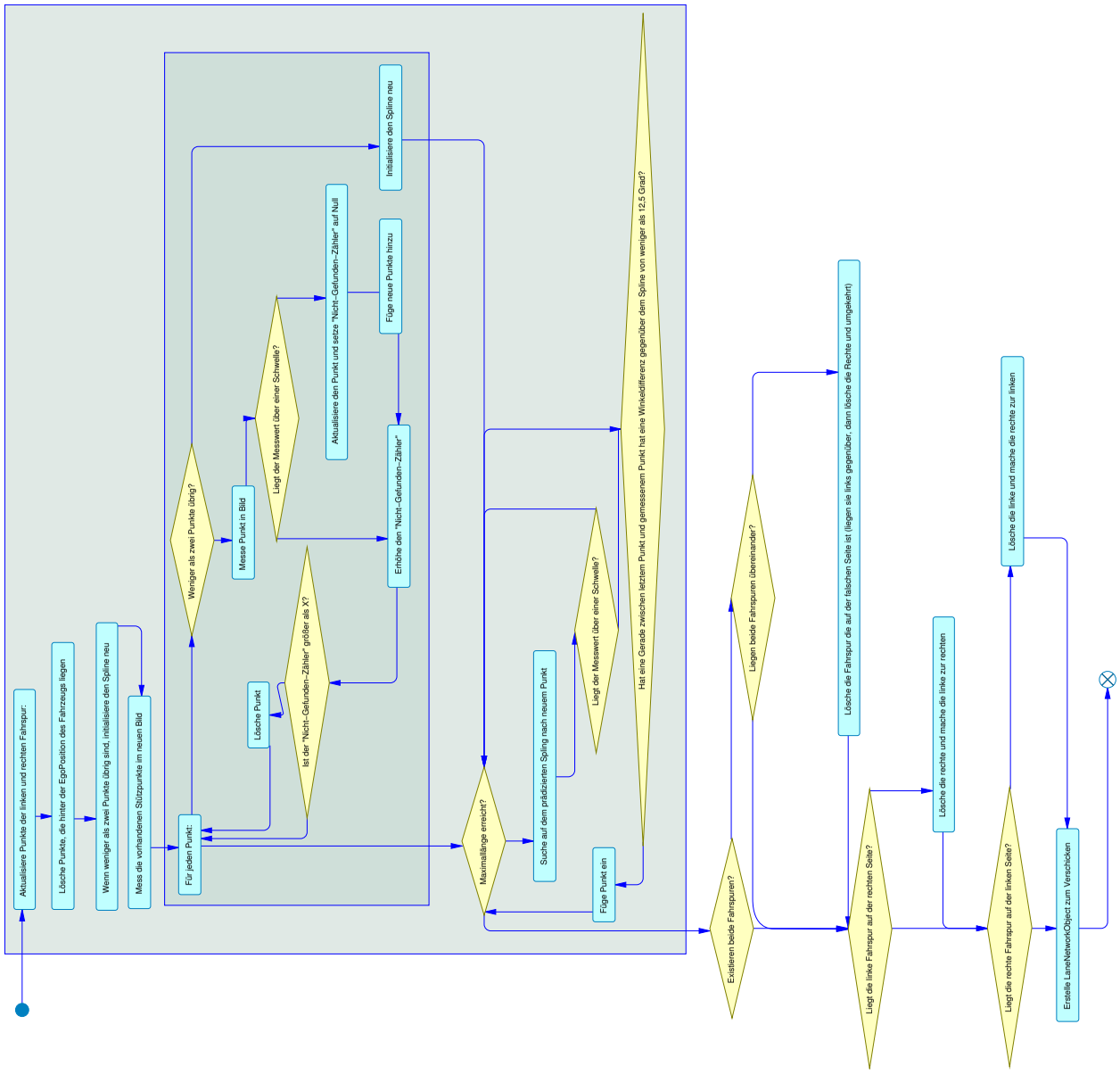


Abbildung 3.2: Tracking zweier Fahrspuren

```
clearPoints();
if (existsOtherLane) {
 insertOtherSpline(*ks);
 measurePoints(newimage);
} else {
 insertLinearPoints(); measurePoints(newimage); rememberPointCount;
 insertLeftCurvePoints(); measurePoints(newimage); rememberPointCount;
 insertRightCurvePoints(); measurePoints(newimage); rememberPointCount;

 insertFigureWithMaxPoints();
}
if (pointCount < 2) clearPoints();
```

Listing 3.1: Initialisierung einer Fahrspur

im Abstand von einem Meter werden neue Punkte gemessen. Ein Punkt wird als neuer Stützpunkt in den Spline übernommen, wenn seine Qualität einen Schwellwert überschreitet und der Winkel zwischen Spline und Messpunkt weniger als 12.5° beträgt. Dieser Wert hat sich bei Versuchsfahrten als sinnvolle Größe ergeben, mit der Kurven noch sicher verfolgt werden können.

Ist die maximale Länge der Fahrspur erreicht oder werden keine sinnvollen Messpunkte mehr erkannt, so führt der Algorithmus noch Verwaltungstätigkeiten durch. Zunächst wird überprüft, ob beide Fahrspuren übereinander liegen. In diesem Fall wird eine der beiden Fahrspuren gelöscht. Da es für die Initialisierung wichtig ist, ob eine Fahrspur links oder rechts liegt, wird überprüft, ob diese Eigenschaft korrekt ist. Bei Fahrspurwechseln kann es vorkommen, dass die linke oder rechte Fahrspur die Seite wechselt. In diesem Fall werden sie getauscht und die jeweils andere gelöscht.

Das hier vorgestellte Verfahren umfasst auf recht einfache Weise die Schritte des Regelkreises aus Abbildung 2.11 sowie die Logik zum Aufsetzen, Verfolgen und Terminieren von Tracks, in diesem Fall Fahrspuren. Der große Vorteil von ortsfesten Objekten ist, dass ihre einzige Bewegung im Messrauschen begründet ist. Regeln und Filter sind so einfach zu handhaben und umzusetzen.

4 Tracking

Als Tracking wird in dieser Arbeit die Verfolgung von Objekten verstanden. Abschnitt 2.5 beschreibt die Grundlagen des Trackings von ortsfesten und dynamischen Objekten. Wie in Abbildung 2.11 dargestellt, besteht der Vorgang des Trackings aus den Phasen Messwerterfassung, Prädiktion und Aktualisierung des Zustandsvektors. Mit Hilfe dieser drei Phasen läßt sich ein einzelnes Objekt verfolgen, von dem exakte Messdaten bestimmbar sind. Das Tracking der Position eines anderen Verkehrsteilnehmers läßt sich beispielsweise mit diesem Vorgehen durchführen. Mit Hilfe eines Abstandsmessgeräts läßt sich ein Positionswert bestimmen. Dieser kann in den Kreislauf eingebracht werden.

Die in dieser Arbeit durchgeführte Umfelderkennung verarbeitet jedoch mehrere Objekte. Dabei muss sie mit neuen Objekten umgehen oder den Verlust von Messobjekten verkraften können. So können beispielsweise neue Tracks entstehen wenn ein Objekt sich in den oder bekannte Objekt sich aus dem Sichtbereich des Sensors bewegt. Durch Messfehler können ebenfalls kurzzeitig Messobjekte entstehen, die aber keine lange Lebensdauer haben und auch keine Repräsentation in der Realität besitzen. Diese werden Sensorgeister genannt.

In den folgenden Abschnitten wird beschrieben, wie die hier entwickelte Software mit Sensorgeistern und realen Objekten umgeht, um ein möglichst stabiles Bild der Umgebung mit Hilfe der vorhandenen Sensoren zu erzeugen.

4.1 Trackinitialisierung

Nicht jedes Sensorobjekt, das ein Sensor erkennt, hat ein Objekt in der realen Welt. Viele Objekte sind Sensorgeister. Das Verfolgen von Geistern sorgt für ein falsches Bild der Welt und erhöht den Rechenaufwand, da mehr Filter gerechnet werden müssen. So kann der Radarsensor zum Beispiel hervorragend Senkkastendeckel im Boden erkennen. Diese Objekte sind zwar tatsächlich vorhanden, haben aber, da sie eben im Boden eingelassen sind, für die künstliche Intelligenz keine Bedeutung. Trotzdem würde das Fahrzeug für diesen Deckel bremsen und ausweichen. Eine andere Art von Geistern sind erkannte Objekte, die durch

Reflexionen zustande kommen. Diese haben im Vergleich zu Senkkastendeckeln keine reale Repräsentation.

Aufgabe der Trackinitialisierung ist es, reale plausible Objekte von nicht realen oder nicht relevanten Objekten zu unterscheiden und nur für diese ein reguläres Tracking aufzusetzen. Ferner wird in dieser Phase versucht, fehlende Messdaten der Objekte zu berechnen. Dies wird zum Beispiel für die Geschwindigkeitsinformation des Hella Sensors gemacht. Eine weitere zentrale Aufgabe ist es, sinnvolle Anfangswerte für den im späteren Tracking verwendeten Kalmanfilter zu bestimmen, da die Anfangswerte maßgeblich für das Einschwingverhalten relevant sind.

Die Trackinitialisierung dieser Software besteht aus fünf Stufen:

- Wiedererkennen eines Objekts an Hand der sensorspezifischen ID
- Zuordnung eines neuen Objekts zu einem bereits bekannten Objekt
- Hinzufügen eines neuen Objekts, das keinem bekannten zugeordnet werden konnte
- Erstellen eines regulären Tracks bei entsprechender Güte eines Objekts
- Terminierung von potenziellen Tracks

Wiedererkennen eines Objekts an Hand seiner ID

Ist ein Sensorobjekt bereits in der Stufe der Initialisierung bekannt, so ist seine ID, die ein Objekt für einen Sensor eindeutig zu einem Zeitpunkt festlegt, spezifiziert. Diese ID wird durch einen HashIndex (siehe Abschnitt 5.3.9) in der Datenbank der potenziellen Tracks gesucht. Wird ein Objekt wiedererkannt, so wird ein Gating durchgeführt. Dieses sorgt dafür, dass kein falsches Objekt für die Aktualisierung des Daten herangezogen wird. Dies kann geschehen, wenn ein Sensor eine ID neu vergibt, was durch recht geringe Datentypgrössen (i.d.R. 8 Bit) oft geschieht.

Ist ein Objekt wiedererkannt, so wird ein einfacher Kalmanfilter mit einem linearen Bewegungsmodell genutzt, um die Objektdaten zu aktualisieren. Dies ist nötig, um für die späteren regulären Tracks sinnvolle verifizierte Daten zu besitzen und damit ein schnelles Einschwingen zu garantieren.

Wird ein Objekt nicht erkannt oder passen die neuen Daten nicht zu den bereits bekannten, so wird versucht, dieses anderen bereits bekannten Objekten zuzuordnen.

Zuordnung eines Objekts zu einem bereits bekannten Objekt

Wird ein Objekt zum ersten Mal erkannt, wird versucht, dieses anderen bereits bekannten Objekten aus der Datenbank der potenziellen Tracks zuzuordnen. Dies ist nötig, um einerseits auf ID-Wechsel innerhalb eines Sensors reagieren zu können, andererseits aber auch, um in Überschneidungsbereichen von zwei Sensoren ein Objekt auch nur als eines und nicht als zwei wahrzunehmen. Diese Mehrfachzuordung ist auch ein zentraler Punkt in der Erstellung eines regulären Tracks, da über sie Plausibilitätsprüfungen durchgeführt werden können.

Eine Zuordnung wird über eine Kostenfunktion bestimmt. Ziel ist das bereits bekannte Objekt zu finden, das die geringsten Kosten hat. Die Kostenfunktion lautet:

$$cost = |x_{neu} - x_{bekannt}| + |y_{neu} - y_{bekannt}| + |vx_{neu} - vx_{bekannt}| + |vy_{neu} - vx_{bekannt}|$$

Mit dieser Funktion lassen sich alle Kerngrößen einer Objektzuordnung gewichten. Zusätzlich zu der Kostenfunktion wird noch ein Gating (siehe Abschnitt 4.1.1) eingesetzt, so dass unplausible Zuordnungen auch bei den geringsten Kosten nicht stattfinden.

Ist diese Zuordnung erfolgreich, so wird das bekannte Objekt mit den Daten des neuen Objekts über einen Kalmanfilter aktualisiert.

Hinzufügen eines neuen Objekts, das keinem anderen zugeordnet werden konnte

Wurde ein Objekt nicht wiedererkannt oder einem bereits bekannten nicht zugeordnet, so werden seine Daten in die Objektdatenbank der potenziellen Tracks übernommen. Hierbei werden nur Daten übernommen, die von dem Sensor, der das Objekt geliefert hat, auch bestimmbar sind. Andere Daten, wie zum Beispiel Geschwindigkeiten, werden bei späteren Aktualisierungen durch einen Kalmanfilter bestimmt.

Erstellen eines regulären Tracks

Erreicht ein bekanntes Objekt eine bestimmte Güte, so wird es von einem potenziellen zu einem regulären Track.

Die Güte eines potenziellen Tracks definiert sich durch die Anzahl an Wiedererkennungen, die bereits erfolgt sind. Mit jeder Wiedererkennung wird ein Zähler erhöht. Wird ein potenzieller Track bei einem Sensordurchlauf nicht aktualisiert, so wird der Zähler dekrementiert.

Überschreitet ein Track eine Schwelle, so werden seine Daten in die Datenbank der regulären Tracks übernommen und aus der Datenbank der potenziellen Tracks entfernt.

Zur Bestimmung der für einen potenziellen Track nötigen Anzahl an Wiedererkennungen wurden experimentell Bereiche bestimmt, welche die Anzahl modifizieren können und nur bei bestimmten Bedingungen gültig sind. Ein Bereich ist definiert durch ein geschlossenes Polygon, eine Zahl, die die Modifikation der Anzahl der Wiederkennungen bestimmt sowie einen Ausdruck in einer domänenspezifischen Sprache (DSL)[MHS05], der die Gültigkeit eines Polygons für einen potenziellen Track bestimmt.

Die Gültigkeit eines Bereichs wird durch eine DSL beschrieben. Diese ist an zweiwertige Logik angelehnt und bietet die Möglichkeit, effizient Kombinationen von Sensortypen zu begünstigen oder zu verschlechtern. Hierzu wurde ein Interpreter entwickelt, der in der Lage ist, eine Zeichenkette einzulesen und dann auszuwerten. Die Lösung der Beschreibung von Sensorkombinationen auf diese Weise bietet die Möglichkeit, effizient die Güte für bestimmte Bereiche ohne eine Neuübersetzung der Software zu ändern. Dies erleichtert den Abstimmungsaufwand erheblich. Listing 4.1 beschreibt die Definition der Sprache. Wie zu sehen ist, behandelt sie die klassischen logischen sowie variablen Elemente zu Auswertung der in einem potenziellen Track vorhandenen Sensortypen.

Die Anzahl bestimmt sich so zu:

$$count = 5 + \sum_{i \in Bereiche} \{Wert_i | pos_{Objekt} \in Polygon_i \wedge Ausdruck_i(Objekt)\}$$

Auf diese Weise lassen sich flexibel Bereiche für bestimmte Sensoren als „verboten" oder als „prädestiniert" markieren, in dem die Zahl auf einen sehr großen oder sehr kleinen Wert gesetzt wird und ein entsprechender Ausdruck als Bedingung angegeben wird. Zum Beispiel

```
TERM       = "( " ATOM " " OP " " ATOM " )" ;
OP         = "&&" | "||" ;
ATOM       = BOOL | SENSORTYPE | NOT | TERM ;
NOT        = "!" TERM ;
BOOL       = "1" | "0" ;
SENSORTYPE = "NoSensor" | "IBEOFront" | "IBEORear" | "IDISFront" | "
   IDISRear" | "SMSFront" | "SMSRear" | "TestSensor1" | "TestSensor2" |
    "MaxValue" ;
```

Listing 4.1: Erweiterte Backus-Naur-Form [ebn96] der DSL zur Gültigkeit von Sensorbereichen

erzeugt der Radarsensor gelegentlich Sensorgeister neben dem Fahrzeug. Ein solches Objekt liegt weit außerhalb seines Sichtbereiches. Ein Polygon, welches diesen Bereich einschließt, einen sehr großen Modifikationswert enthält und einen Ausdruck der Form „(SMSFront && 1)" erhält, behebt dieses Problem. Eine weitere Anwendung ist das Vermeiden von Sensorgeistern vor dem Fahrzeug. Geister dieser Art treten häufig durch Nickbewegungen auf. Dies führt dazu, dass Sensoren den Boden als Hindernis wahrnehmen. Diese „Bodengeister" können durch drei Polygone im Überlappungsbereich der vorderen Sensoren mit einem hohen Wert und Permutationen der Bedingung „(SMSFront && !(IBEOFront || IDISFront))" angegangen werden. Auf diese Weise wird ein Objekt, das nur von einem Sensor erkannt wird, nicht zu einem regulären Track. Das ist erwünscht, da diese Bedingungen nur die Überlappungsbereiche betreffen und in diesen beide Sensoren das Objekt wahrnehmen müssten, wenn es denn wirklich dort wäre.

In Listing 4.2 ist ein Beispiel der beschriebenen DSL angegeben. Es beschreibt den redundanten Sichtbereich des vorderen Infratorsensors. Der Sichtbereich wird durch das Polygon beschrieben (gleichschenkliche Dreieick mit 30m Lotlänge, beginnend im Ursprung des Fahrzeugs). Der angegebene modifyCount beschreibt die Anzahl der Wiedererkennungen, die nötig sind, damit ein Sensorobjekt in diesem Bereich zu einem Track wird. Der Wert 2000 resultiert aus der Annahme, dass der Versuchsträger nicht lange genug an einem Ort steht, um ein Objekt 2000 mal zu erkennen. In der condition ist ein Ausdruck abgelegt, der der beschreibenen DSL entspricht. Er beschreibt den Fall, dass dieser Bereich nur aktiv wird, wenn das Objekt nur von einem Sensor erkannt wurde (IDIS und nicht ibeo oder sms). Auf diese Weise werden Single-Sensor-Objekte, die nur vom Infrarotsensor erkannt werden, in der Realität nie in das Tracking übernommen.

Terminierung von potenziellen Tracks

Potenzielle Tracks, die einmal erstellt worden sind und deren Güte nicht hoch genug ist, um ein regulärer Track zu werden, müssen wieder aus dem System entfernt werden. Dies geschieht über die beschriebenen Wiedererkennungsanzahlen. Am Ende eines jeden Sensordurchlaufs

```
...
polygon={0,0;30,2.9;29,-2.9;0.0}
modifyCount=2000
condition=( IDISFront && !( IBEOFront || SMSFront ) )
...
```

Listing 4.2: Ausschnitt aus der Konfigurationsdatei der Sensorsichtbereiche

werden für jeden nicht aktualisierten potenziellen Sensortrack eines Sensortyps die Zähler dekrementiert. Dies geschieht solange, bis sie den Wert Null erreichen. In diesem Fall werden sie aus dem System entfernt.

4.1.1 Gating in der Trackinitialisierung

Das Gating, das in den fünf Schritten eingesetzt wird, betrachtet vier Messgrößen: X-Position, Y-Position, Geschwindigkeit in X-Richtung und Geschwindigkeit in Y-Richtung. Die Formel ergibt sich so zu:

$$
\begin{aligned}
&Gating = \\
&\quad (|x_{neu} - x_{bekannt}| < xthres \wedge |y_{neu} - y_{bekannt}| < ythres \wedge \\
&\quad |vx_{neu} - vx_{bekannt}| < vxthres \wedge |vy_{neu} - vx_{bekannt}| < vythres)
\end{aligned}
$$

Die Schwellwerte werden zunächst mit einem Startwert initialisiert und nach jedem Sensordurchlauf über ein Exponential-Backoff [Wik07b] angepasst. Danach wird die mittlere Abweichung der Werte bestimmt und diese mit den Schwellwerten verglichen. Überschreitet die mittlere Abweichung den Schwellwert um mehr als 10%, so wird er entsprechend angehoben. Bei einer Unterschreitung um 5% wird er entsprechend erniedrigt, wobei er einen Wert von von 1 bei V_x, V_y und 2 bei X, Y nicht unterschreiten darf. Die Werte haben sich bei Experimenten als sinnvolle Größen herausgestellt, die Fehlzuordnungen vermeiden und dennoch richtige Zuordnungen erlauben.

Um das Gating und auch die oben beschriebene Kostenfunktion nutzen zu können, muss ein Objekt auf einen Punkt reduziert werden. In Abschnitt 2.5 wurde bereits darauf eingegangen, dass Objekte in diesem Projekt beliebige Formen besitzen können.

Um dennoch einen einzigen Punkt auswerten zu können, wird die Annahme gemacht, dass ein Objekt an dem zum Fahrzeug nächsten Punkt von einem Sensor am stabilsten erkannt wird. Listing 4.3 beschreibt ein Verfahren, mit dem ein solcher Punkt gefunden wird. Dazu wird zunächst der minimale Abstand des Fahrzeugs zu der durch die Konturpunkte beschriebenen Geraden bestimmt und dazu ein Lotpunkt errechnet. Dieser wird durch den Hüllquader des Objekts gedeckelt. So wird eine relative Stabilität des Punktes erreicht. Dieser kann sich mit dem Fahrzeug mitbewegen. Der Fall tritt aber nur bei sehr großen, „langen" Objekten (z.B. Wänden) auf. Lange Objekte werden allerdings auf Grund ihrer großen Fläche dennoch korrekt erkannt.

```
for every point in contour do
{
 calculate distance from car to line between this and previous point;
 if (dist < mindist)
 {
  calculate solder point;
  set point to solder point;
  if (solder not in rectangle of contour)
  {
   clamp point to rectangle of contour
  }
 }
}
return point;
```

Listing 4.3: Bestimmung eines verfolgbaren Punktes

4.2 Objectmerging

Das Zusammenfügen (engl. merging) von mehreren Sensorobjekten zu einem Track ist eine zentrale Aufgabe der Fusion. Sie sorgt dafür, dass Tracks entstehen können, die von mehreren Sensoren erkannt wurden und in der Fusion trotzdem als ein und derselbe Track gehandhabt werden. Ferner erkennen die Sensoren teilweise ein und dasselbe Objekt mehrfach an unterschiedlichen Stellen. So sieht der Radarsensor eine Wand als mehrere Punktobjekte, der Laserscanner diese aber als ein langes, lininenförmiges Objekt. Auch hier soll nur ein einziger Track verfolgt werden.

Merging von Sensordaten und Tracks

Um dies zu erreichen, wird im Vergleich zur Trackinitialisierung keine feste Zuordnung von Sensorobjekt-IDs zu Tracks durchgeführt. Ein Track existiert nach seiner Erzeugung durch die Initialisierung für sich selbst, bis er aufgespalten (siehe Abschnitt 4.4) oder terminiert (siehe Abschnitt 4.5) wird. Um eine Aktualisierung eines Tracks durch ein Sensorobjekt durchzuführen, wird für jedes Sensorobjekt der optimale Track durch eine Kostenfunktion gefunden. Durch diese Zuordnung können zu einem Track auch mehrere Sensorobjekte gefunden werden, für die ein Track optimal ist. Dies führt eventuell zu mehreren Updatevorgängen (siehe Abschnitt 4.3). Bei einer korrekten Zuordnung erhöht es zusätzlich die Sicherheit.

Drei Klassen von Objekten treten in der Software auf:

- Punktobjekte (1-Punkt Objekte)
- Linienobjekte (2-Punkt-Objekte)
- Polygonobjekte (N-Punkt-Objekte $N > 2$)

Daraus ergeben sich 12 Kombinationsmöglichkeiten zur Bestimmung der Kostenfunktion für die Zuordnung. Diese wurden auf vier Kostenfunktionen reduziert, die alle Kombinationen abdecken.

- Punkt zu Punkt:

$$cost = ||\overrightarrow{Point_{left}} - \overrightarrow{Point_{right}}||_2$$

- Punkt zu Linie:

$$cost = min\{||\overrightarrow{p_1p_2} - \overrightarrow{Point}||_2, ||\overrightarrow{p_1} - \overrightarrow{Point}|_2, ||\overrightarrow{p_2} - \overrightarrow{Point}||_2\}$$

- Polygon zu Punkt :

$$cost = min\{||\overrightarrow{p} - \overrightarrow{Point}||_2 \forall p \in Polygon\}$$

- Polygon zu Polygon Variante 1 [ope07] :

$$cost = \sum_{i=1}^{7} |sign(h_i^{Polygon1}) * \log(h_i^{Polygon1}) - sign(h_i^{Polygon2}) * \log(h_i^{Polygon2})| * 10 + |len(Polygon1) - len(Polygon2)|, h_i^x \text{i-tes Hu-Moment des Polygons x}$$

- Polygon zu Polygon Variante 2 :

$$cost = avg(min\{||\overrightarrow{p} - \overrightarrow{P}||_2 \forall p \in Polygon1\} \forall P \in Polygon2)$$

Für die Kostenfunktion Punkt zu Punkt wurde der euklidische Abstand gewählt. Dieser lässt sich effizient berechnen und ist durch seinen physikalischen Bezug gut zu verstehen. Für Punkt zu Linie wurde ebenfalls der euklidische Abstand gewählt. Hier wird zwischen einem Punkt unterschieden, dessen Lotpunkt zwischen den beschreibenden Punkten der Linie liegt, und einem, der außerhalb liegt. Liegt der Punkt außerhalb, so wird der minimale Abstand zu einem der beiden Randpunkte der Linie als Kostenfunktion genutzt. Liegt er zwischen den Punkten, so wird der Abstand Punkt-Linie berechnet. Im Fall Polygon zu Linie wird der minimale euklidische Abstand des Punkts zu den Punkten des Polygons als Wert genutzt.

Für die Variante Polygon zu Polygon wurden zwei Ansätze getestet. Die Variante 1 nutzt Hu-Momente [Hu62]. Das sind sieben Bildinvarianten, die ein Bild unabhängig von Eigenschaften wie Ausrichtung und Größe identifizieren. Diese Momente werden für beide Polygo-

ne berechnet und über die oben angegebene Funktion ein Ähnlichkeitswert berechnet. Um Informationen über die Ausdehnung der Polygonzüge ebenfalls in die Kosten eingehen zu lassen, wurde zusätzlich die Differenz der Polygonlängen hinzu addiert. Das Ergebnis ist ein Kostenwert, der eindeutig ist, aber dessen Wert nur einen geringen Bezug zu den anderen Kostenfunktionen hat. Dies ist jedoch nötig, da alle vier Funktionen miteinander vergleichbar seien müssen, um ein Minimum zu bestimmen. Aus diesem Grund wurde die Variante 2 entwickelt, die den durchschnittlichen minimalen Abstand jedes Punktes zu jedem anderen errechnet. Dieser Wert ist mit den anderen Funktionen gut vergleichbar, da er ebenfalls auf dem euklidischen Abstand fußt.

Zusätzlich zu dem hier geforderten Minimum der Kosten einer Zuordnung wird ein Gating der Zuordnungen durchgeführt. Dies ist nötig, damit eine sehr kostenintensive Zuordnung nicht durchgeführt wird, obwohl es die minimalen Kosten sind. Ist beispielsweise ein Objekt 20 Meter von einem anderen entfernt und kein anderes Objekt ist näher als 20 Meter an dem gemessenen Objekt gelegen, so darf es nicht zugeordnet werden, obwohl es das Minimum darstellt. In Abhängigkeit des Sensortyps werden unterschiedliche Maximalwerte angenommen. Dies ist nötig, da die Radarsensoren teilweise eine nicht sehr gute Positionsbestimmung bieten. Sie geben zwar die Existenz eines Objekts an, liegen aber gegenüber den anderen Sensoren oft erheblich neben der exakten Position. Aus diesem Grund ist eine Zuordnung von Radarobjekten nur bis zu maximalen Kosten von einem Meter möglich. Für die anderen Sensortypen hat sich ein Maximum von drei Metern als sinnvoll erwiesen.

Merging von Tracks mit Tracks

Treten Sensorobjekte zeitgleich auf, werden sie, wie in Abschnitt 4.1 beschrieben, bis zur Erstellung eines regulären Tracks nur anhand ihrer ID unterschieden. Aus diesem Grund kann es geschehen, dass Objekte, obwohl sie sehr nahe beieinander liegen, als zwei Tracks verwaltet werden. Eine weitere Möglichkeit ist, dass ein bewegtes Objekt sich auf ein statisches zubewegt und dort zum Stehen kommt. Die eben beschriebene Zuordnung würde stets eine Zuordnung mit minimalen Kosten finden und so die Objekte nicht verschmelzen. Dies hat neben dem erhöhten Verwaltungsaufwand für die Sensor-Daten-Fusion den Nachteil, dass auch die KI sich mit Objekten beschäftigen muss, die so nahe beieinander sind, dass diese Information für sie nicht relevant ist. Ein weiterer Anwendungsfall, bei dem oft sehr nahe Objekte entstehen, sind Bereiche mit hohem Rauschen, die beispielsweise an den Sichträndern oder auf Grünflächen auftreten. Hier werden oft viele Objekte erzeugt und wieder vernichtet, da sie nicht lang genug Bestand haben.

Um diese Probleme anzugehen, wird eine zusätzliche durch Zeit ausgelöste Mergingfunktion entwickelt. Diese arbeitet mit der in Abschnitt 4.2 beschrieben Kostenfunktion, vergleicht jedoch Sensortracks miteinander und nicht Tracks mit Sensorobjekten. Ferner wird ein sehr viel härteres Gating angesetzt. Dieses beginnt erst ab einem halben Meter Abstand, Objekte zu verschmelzen.

Listing 4.4 beschreibt den Algorithmus. Das Merging von Tracks vergleicht jeden mit jedem anderen Track, der der Sensor-Daten-Fusion bekannt ist. Ist die Kostenfunktion innerhalb des Gatings, so werden die Tracks verschmolzen. In diesem Fall werden die Konturpunkte beider Tracks zu einem zusammengefasst und einer der Tracks gelöscht. Schließlich wird zum nächsten Track übergegangen. Da diese Funktion in regelmäßigen Abständen aufgerufen wird, ist ein rekursiver Aufruf, um mehr als zwei Tracks auf einmal zu verschmelzen, nicht nötig. Wäre es beispielsweise nötig, drei Tracks miteinander zu mergen, so würden im ersten Durchlauf zunächst zwei der Tracks miteinander verschmolzen. Einen Timeslot später würde der dritte Track hinzukommen.

4.3 Trackupdate

Im Abschnitt 4.2 wurde beschrieben, wie einem Track ein oder mehrere Sensorobjekte zugeordnet werden. Diese Zuordnung wird genutzt, um eine Aktualisierung von Tracks durchzuführen. Dazu wird der in Abschnitt 2.5 beschriebene Kalmanfilter genutzt. Dieser läßt sich nicht direkt auf das Problem anwenden, da ein Sensorobjekt und auch ein Track nicht aus einer Position, sondern aus einem Polygon mit einem oder mehreren Punkten besteht. Es wird die Annahme getroffen, dass sich ein Track nur als Gesamtobjekt in der Welt bewegt, die Parameter wie Geschwindigkeit oder Beschleunigung also für jeden Stützpunkt des Polygons gleich sind. Ferner wird für die Position nur ein Varianzwert mitgeführt. Diese Annahme reduziert die Rechenzeit erheblich, da die einzelnen Punkte unabhängig von den anderen Parametern gefiltert werden können. Der Nachteil dieser Annahme besteht in dem Verlust der Möglichkeit, partielle Änderungen der Kontur korrekt zu tracken. Diese Einschränkung stellt keinen großen Verlust dar, da Objekte großer Ausdehnung meist ortsfest sind und bei kleinen Objekten kein Raum für partielle Änderungen vorhanden ist. Listing 4.5 beschreibt den Ablauf der Aktualisierung von Tracks. Er spaltet sich in die Phasen Stützpunktzuordnung, Stützpunktaktualisierung und Stützpunkterstellung. Sie werden nun näher beschrieben.

```
mergeMapping = ();
for every track1 = track in database {
 if (track1 in mergeMapping) continue;
 for every track2 = track in database {
  if (track1 == track2 || track2 in mergeMapping) continue;
  if (cost(track1,track2) < threshold) {
   put (track1,track2) in mergeMapping;
  }
 }
}
for every mergePair in mergeMapping {
 merge mergePair in database;
}
```

Listing 4.4: Merging von Tracks mit Tracks

```
calculateBestPointMatching();
predictTrack();
forevery matched Point
{
  correctionVector = gain*(predictedPos-measuredPos);
  newPos = predictedPos+correctionVector;
  correctionSumVector += correctionVector;
}
newState = predictedState + correctionVector/count of matched Points;
addNotMatchedPoints();
```

Listing 4.5: Aktualisierung eines Tracks

4.3.1 Stützpunktzuordnung

Um eine derartige Aktualisierungsmethode nutzen zu können, muss zunächst eine Zuordnung der Stützpunkte des Objekts zu den Stützpunkten des Tracks durchgeführt werden. Zu diesem Zweck wurden zwei Methoden getestet:

- Munkres-Algorithmus [FWM94]: Diese Methode ordnet jedem Punkt einen anderen Punkt zu, basierend auf einer Kostenfunktion. Hierbei wird mit einer Komplexität von $O(AnzahlObjektpunkte^2 * AnzahlTrackpunkte)$ immer das globale Minimum der Zuordnung gefunden. Dabei lassen sich auch Zuordnungen verbieten. Das ist nötig, damit keine Zuordnungen durchgeführt werden, die weit außerhalb des Gatingbereiches liegen, obwohl sie das globale Minimum darstellen.
- Minimum-Filter [Eff02]: Der Minimumfilter findet im Vergleich zum Munkres nicht das globale sondern nur ein lokales Minimum bezüglich der Kostenfunktion. Dieser besitzt eine Komplexität von $O(AnzahlObjektpunkte * AnzahlTrackpunkte)$ und ist deutlich einfacher und schneller als der Munkres.

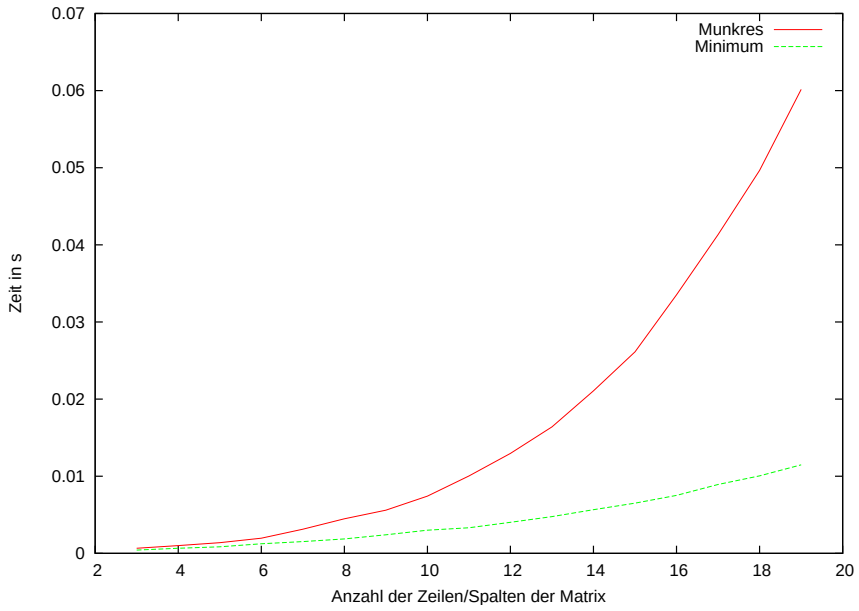


Abbildung 4.1: Vergleich des Minimum- mit dem Munkres-Algorithmus

Auf Grund der Komplexität und der Rechenzeit (siehe Abbildung 4.1) wird in diesem Projekt der Minimumfilter eingesetzt. Dieser liefert für eine Zuordnung der Punkte eine ausreichende Genauigkeit. Als Kostenfunktion wird der euklidische Abstand gewählt. Dieser wird mit einem Gate von vier Metern versehen, ab welchem eine Zuordnung ausgeschlossen wird.

4.3.2 Stützpunktaktualisierung

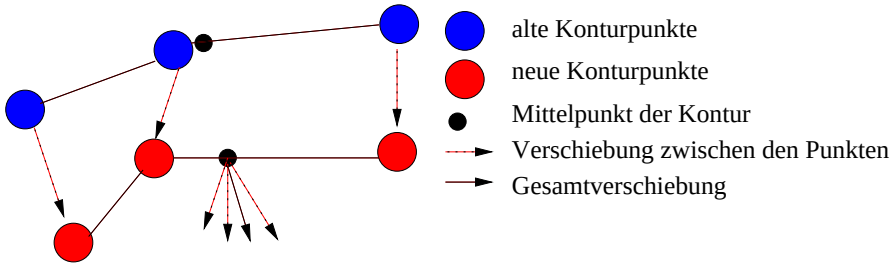


Abbildung 4.2: Bestimmung der Verschiebung der Kontur eines Objektes

Um die in in Abschnitt 4.3.1 gefundene Zuordnung verwenden zu können, wird zunächst der Prädiktionsschritt des erweiterten Kalmanfilters (siehe Abschnitt 2.5) berechnet. Dadurch werden die bekannten Stützpunkte entsprechend der Zustandsgrössen verschoben. Anschließend wird für jeden zugeordneten Punkt der Innovationsvektor (siehe Abbildung 4.2) bestimmt und mit dem Kalman Gain multipliziert (siehe Listing 4.5). Auf diese Weise wird die neue gefilterte Position des Punktes bestimmt. Die Differenz der neuen zur prädizierten Position wird aufsummiert, nach der Verarbeitung aller Punkte durch die Anzahl geteilt und zur Aktualisierung der anderen Zustandsgrössen genutzt.

4.3.3 Stützpunkterstellung

Nicht zugeordnete Stützpunkte werden dem Track hinzugefügt. Da die Menge der Stützpunkte einen offenen Polygonzug beschreibt, ist die Reihenfolge der Punkte relevant. Um eine stabile Sortierung zu gewährleisten, wurden zwei Algorithmen entwickelt.

Der in Listeing 4.6 beschriebene Algorithmus optimiert das Polygon auf eine minimale Gesamtlänge. Zunächst wird der Abstand zwischen zwei bereits im Polygon vorhandenen Punkten sowie der Abstand einer der Punkte zu dem neu einzufügenden Punkt bestimmt. Danach wird mit der Formel „alte Gesamtlänge - Abstand zwischen den beiden vorhanden Punkten + Abstand erster alter Punkt zu neuem Punkt + Abstand nächster alter Punkt zu neuem Punkt" das Minimum gesucht. An dieser Stelle wird schließlich der neue Punkt eingefügt. Dieser Algorithmus zum Einfügen eines Punktes hat eine Komplexität von $O(n)$, multipliziert mit der Anzahl der einzufügenden Punkte.

Der durch Listing 4.7 beschriebene Algorithmus sortiert die Punkte von rechts nach links entsprechend der Ergebnisse der atan-Funktion unter Beachtung der Quadranten an. Die

```
listofNewDists = ();
listofChainDists = ();
for every point in track
 {
  if (point != lastPoint)
  {
   sum += dist(point,nextPoint);
   push dist to listofChainDists;
  }
  push dist(point,newPoint) to listofNewDists;
 }
for every chainDist in listofChainDists and every newDist in
   listofNewDists
 {
   tmp = sum - chainDist + newDist + nextNewDist;
   if (tmp < minimalDist)
   {
    minimalDist = tmp;
   }
 }
insert newPoint at minimalDist;
```

Listing 4.6: Bestimmung eines verfolgbaren Punktes

```
for every point in track
 {
  calculate angle between point and ego position;
 }
quick sort list of points by angle;
```

Listing 4.7: Sortierung eines Konturpolygonzuges

Komplexität des Algorithmus wird durch den Quick Sort Algorithmus vorgegeben und bestimmt sich so zu $O(n * \log(n))$.

Beide Algorithmen liefern gute Ergebnisse. Der erste Algorithmus hat jedoch Probleme mit dem Handling von Randpunkten und großen Lücken in den Konturen. So werden Punkte, die einen großen Abstand zu allen anderen Punkten haben, an den Enden einsortiert. Dieses führt zu nicht der Realität entsprechenden Konturzügen. Der zweite Algorithmus wird in der hier beschriebenen Software eingesetzt. Er besitzt die geringere Laufzeit und zeichnet sich durch seine Nähe zur Arbeitsweise der verwendeten Sensoren aus.

4.3.4 Stützpunktterminierung

Wird ein Stützpunkt längere Zeit nicht aktualisiert, so muss er aus der Menge entfernt werden. Dazu führt jeder Stützpunkt einen Zähler mit, der angibt, wie oft er bereits aktualisiert wurde, sowie einen Zeitpunkt, wann dies zuletzt der Fall war. Für die Terminierung eines Punktes gibt es zwei Gründe: Er wurde zu lange nicht aktualisiert oder er wurde seltener aktualisiert, als er erkannt wurde. In beiden Fällen wird er aus der Menge entfernt.

4.3.5 Objektklassifikation

Das Datenformat für das FusionObject (siehe Abschnitt A.3), das an die künstliche Intelligenz übermittelt wird, enthält mehrere Datenfelder, die die Klassifikationen von Objekten beschreiben. Im Rahmen dieser Arbeit werden zwei Klassifikationen umgesetzt. Die Klassifikationen werden nach jeder Trackaktualisierung neu bestimmt.

ObjectRating

Das ObjectRating kann die Zustände notConfirmed, confirmed und multiSensor annehmen. Der Fall, dass ein Track nicht bestätigt wird, kann durch die Verarbeitungsschritte, die zur Erstellung eines Tracks führen, nicht auftreten. Wird ein Track erstellt, so ist er bereits durch die Trackinitialisierung mehrere Male bestätigt worden. Der Unterschied zwischen confirmed und multiSensor leitet sich unmittelbar aus der Anzahl der Sensoren ab, die einen Betrag zu diesem Track leisten. Wird ein Objekt von mehr als einem Sensor gemessen, so erhält der Track das Attribut multiSensor. Im Fall, dass nur ein Sensor ein Objekt erkannt hat, erhält es den Zustand confirmed.

ObjectMovement

Die Klassifikation nach bewegten und unbewegten Objekten ist für die künstliche Intelligenz von großer Bedeutung. So werden unbewegte Objekte beispielsweise überholt, bewegte werden als Folgeobjekt identifiziert und die Geschwindigkeit des Fahrzeugs entsprechend angepasst.

Bei Objekten, die keine Ausdehnung haben (Punkt- oder Linienobjekte), ist diese Entscheidung recht einfach zu treffen. Hier können gemessene oder durch Positionsänderungen bestimmte Geschwindigkeiten zur Klassifikation genutzt werden. Diese Daten werden über das Tracking eines Objekts bestimmt. Objekte, die eine Kontur besitzen, lassen sich auf diese Weise nicht klassifizieren. Bei ihnen ist relevant, welcher Teil eines Objekts von dem Sensor gesehen werden kann. Durch die Änderung der Eigenposition des Fahrzeugs können andere Bereiche eines Objekts sichtbar werden. Fährt das Fahrzeug beispielsweise an einem Fahrzeug vorbei, so errechnet das Tracking eine Geschwindigkeit, die sich aus der Verschiebung des Schwerpunkts des Objekts berechnet. Ein solches Objekt muss jedoch weiterhin als statisches Objekt erkannt werden.

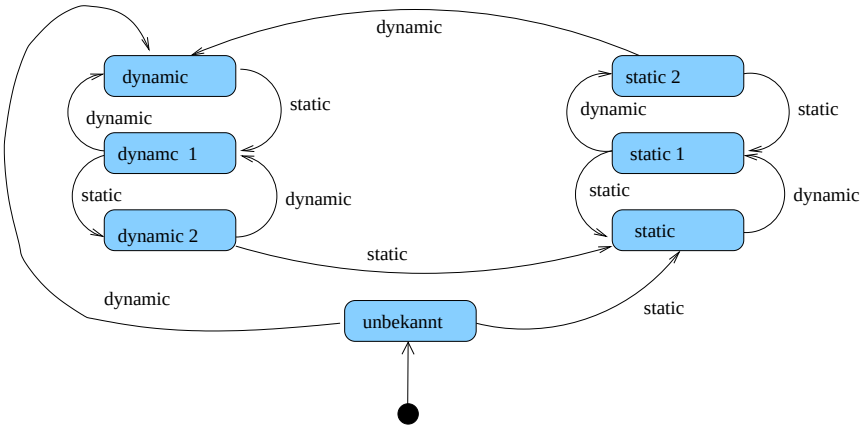


Abbildung 4.3: Gedächtnis der statisch/dynamisch-Entscheidung am Beispiel eines Gedächtnisses mit drei Speicherplätzen.

Um eine Erkennung von statischen und dynamischen Objekten zu gewährleisten, wird in dieser Arbeit die Verschiebung des Hüllquaders betrachtet. Verschiebt sich eine Ecke des Rechtecks nicht, so hat sich das Objekt effektiv nicht bewegt, sondern lediglich erweitert. In diesem Fall wird ein Objekt als statisch angenommen. Bewegt sich ein Objekt mit mehr als $3\frac{m}{s}$ oder mit mehr als 1 m/s und die gemittelte Varianz ist sehr klein, so wird es als dynamisches Objekt angenommen.

Um der Entscheidung eine gewisse Trägheit zu geben, werden die Entscheidungen über einen gewissen Zeitraum gemittelt. Abbildung 4.3 beschreibt diese Trägheit am Beispiel eines Gedächtnisses mit drei Speicherplätzen.

4.4 Objectsplitting

Durch das in Abschnitt 4.2 beschriebene Verfahren zum Zusammenfügen von einzelnen Sensorobjekten eines oder mehrerer Sensoren kann es geschehen, dass unterschiedliche Objekte in der realen Welt zu einem Track zusammengefasst werden. Dieses ist explizit erwünscht, da es für das Fahrzeug ab einer gewissen Nähe keinen Unterschied macht, ob es die Realität als ein oder mehrere Objekte wahrnimmt, außerdem wird die benötigte Rechenleistung reduziert, da weniger Objekte verarbeitet werden müssen.

Die Verschmelzung von Tracks macht ein Verfahren zur Aufspaltung nötig. Es hat die Aufgabe, einen Track, dessen Einzelobjekte sich zu unterschiedlich verhalten bzw. zu weit auseinander bewegen, in mehrere neue Tracks zu teilen.

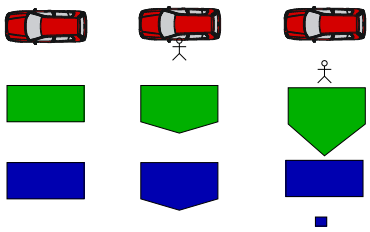


Abbildung 4.4: Aussteigende Person. Realität, Track, Track nach Aufspaltung (von oben nach unten). Person sitzt im Fahrzeug, Person steht neben dem Fahrzeug, Person entfernt sich (von links nach rechts).

Ein Beispiel (siehe Abbildung 4.4) für einen solchen Fall ist ein Insasse, der aus einem Fahrzeug aussteigt. Sitzt der Insasse noch innerhalb des Fahrzeugs, so wird er von den Sensoren als ein Objekt wahrgenommen. Öffnet er die Tür und steht dicht neben dem Fahrzeug, so wird das Objekt etwas größer. Bewegt er sich weg, müssen sich zwei Objekte bilden. Eines beschreibt den Insassen, der sich von dem Fahrzeug entfernt, während das andere das Fahrzeug beschreibt. Findet kein Aufspalten des Tracks statt, so existiert lediglich ein Track, der zwischen dem Insassen und dem Fahrzeug eine „Brücke" aufbaut, die nicht durch Konturpunkte gestützt wird. Da aber sowohl der Insasse als auch das Fahrzeug dauerhaft durch die Sensoren erkannt werden, existiert der Track weiter.

Um dieses Problem zu behandeln, wurde im Rahmen dieser Arbeit ein Algorithmus entwickelt, der das Aufspalten eines Tracks bei zu großem Abstand seiner Einzelobjekte realisiert. Zu diesem Zweck wird die durch den Track beschriebene Fläche als planarer ungerichteter zweigefärbter Graph dargestellt. In diesem Graphen werden wird nach Partitionen gleicher Farbe gesucht. Eine Partition stellt einen neuen, aufgespaltenen Track dar.

Der Algorithmus setzt sich aus folgenden Schritten zusammen:

1. Zunächst wird ein planerer, ungerichter, färbbarer, rechteckiger Graph erstellt, dessen Knoten alle die Farbe Schwarz besitzen.
2. Die einzelnen Polygonzüge aller Sensorobjekte eines Tracks werden in dem Graphen mit einer Breite von drei Knoten weiß eingefärbt.
3. Anschließend werden alle gleichgefärbten unabhängigen Mengen [HMU03] gesucht.
4. Wenn mehr als eine Menge gefunden wurde, werden neue Tracks aus den Punkten, die die Polygonzüge der unabhängigen Mengen beschreiben, erstellt.

Der hier beschriebene Algorithmus besitzt eine Komplexität, die proportional zu seiner Fläche (a) ist ($O(a)$). Dies ist deshalb der Fall, da der Graph im Vergleich zum allgemeinen Fall des NP-vollständigen Problems der Unabhängigen Menge [HMU03] das Attribut planar besitzt. Dies reduziert die Komplexität erheblich, da jeder Knoten nur noch einmal besucht werden muss.

```
bool findIndepentendSet(x,y,set)
{
 return false if (node(x,y) has black color);
 put x,y to set;
 set color of node(x,y) to black;
 call findIndependentSet for every neighbour node;
 return true;
}
setOfSets = ();
set = ();
for every node
{
 if (findIndepentendSet(x,y,set))
 {
   push set to setOfSets;
   set = ();
 }
}
```

Listing 4.8: Suche nach unabhängigen Mengen in einem planaren Graphen

Der hier verwendete Algorithmus (siehe Listing 4.8) ist ein klassischer Backtrackingansatz. Er besitzt normalerweise eine Komplexität von $O(n^n)$. Diese wird durch das schwarze Einfärben eines besuchten Knotens zu $O(n^2)$ bzw. $O(a)$ reduziert, da ein Knoten nicht ein zweites Mal in die Betrachtung mit einbezogen werden kann.

Mit Hilfe von Bibliotheken zur Bildverarbeitung (Verwaltung der Bilder (Graphen), Zeichnen von Polygonen) lässt sich eine effiziente Implementierung der Aufspaltung von Tracks erstellen.

4.5 Trackterminierung

Die Trackterminierung stellt einen sensiblen Bereich des Trackmanagements dar. Die Aufgabe ist es, Tracks, die keine Aktualisierungen mehr von den Sensoren erhalten, aus dem Trackmanagement zu entfernen. Die Herausforderung besteht darin, Tracks nicht zu schnell zu löschen, um nicht eine kurzzeitige Verdeckung des Objekts gleich zum Abreißen des Tracks führen zu lassen und dennoch keine unsinnigen Tracks, die länger nicht aktualisiert wurden, verwalten zu müssen.

Zu diesem Zweck wird der Zeitpunkt der letzten Aktualisierung eines Tracks gespeichert. Wenn dieser bereits zu weit in der Vergangenheit liegt, so wird der Track aus dem Trackmanagement entfernt. Ein weiterer Grund, einen Track zu terminieren, ist das Fehlen von Konturpunkten. Dieser Fall muss allerdings nicht weiter behandelt werden, da er zwangsläufig dazu führt, dass ein Track nicht mehr aktualisiert werden kann und somit kurzfristig gelöscht wird.

5 Architektur

Dieser Teil der Arbeit behandelt zunächst die Architektur des CarOLO Projekts, um eine Einordnung für den Beitrag dieser Arbeit zu bestimmen. Im Anschluss wird die Architektur der Fusion dieser Arbeit detailliert beschrieben sowie auf deren Visualisierung eingegangen.

5.1 Grobarchitektur des Gesamtprojekts

In Abschnitt 1.1 wurde die Aufgabe des Projekts beschrieben. Abbildung 5.1 zeigt die an diesem Problem beteiligten Softwarekomponenten. Der Gesamtstruktur liegt das Pipes and Filters Muster [BMR+96] zugrunde.

Das Pipes and Filters Muster bietet eine Struktur für Systeme, die einen Strom von Daten verarbeiten müssen. Dazu werden Daten über Pipes von einer zur anderen Stufe verschickt und dort von Filtern verändert. Dieses Muster bietet eine optimale Grundlage für dieses Projekt, da große Datenmengen in Echtzeit verarbeitet werden müssen. Jeder der Filter reduziert die Datenmenge so, dass schließlich daraus die richtigen Entscheidungen für die Ansteuerung des Fahrzeugs bestimmt werden können.

Die zentralen Module des Projekts sind Sensorik, Fusion, digitale Karte, künstliche Intelligenz, Aktorik, Fahrspurerkennung und Watchdog. Im Folgenden werden diese Module kurz vorgestellt. Wie in Abbildung 5.1 beschrieben, gibt es noch einige weitere Module. Sie haben mehr zusätzlichen Charakter oder sind für Simulationszwecke nötig. Sie werden deshalb hier nicht vorgestellt.

Sensorik

Das Sensorik Modul liest die anfallenden Daten aus den unter Abschnitt 2.3 beschrieben Sensoren aus. Sie werden durch eine einheitliche Datenstruktur beschrieben, die bereits mit der Eigenposition des Fahrzeugs fusionierte Daten im Weltkoordinatensystem enthält. In

Abschnitt 5.2 wird die Datenstruktur näher erläutert. Die anfallenden Daten werden nun über das Netzwerk an das Fusionsmodul verschickt.

Fusion

Das Fusionsmodul wird in dieser Arbeit beschrieben. Es empfängt die Daten von der Sensorik, fusioniert sie miteinander und verschickt sie an die digitale Karte.

Digitale Karte

Um die von der Fusion produzierten Tracks verarbeiten zu können, wird von der digitalen Karte eine Belegungskarte erzeugt sowie Algorithmen zur Bahnplanung für die künstliche Intelligenz implementiert. Sie fungiert als Dienstleister für die künstliche Intelligenz.

Künstliche Intelligenz

Sie ist das „Gehirn" des Fahrzeugs. Sie trifft aus allen ihr vorliegenden Informationen Entscheidungen über Weg und Geschwindigkeit des Fahrzeugs mit dem Ziel, die gerade aktuelle Mission zu erfüllen.

Aktorik

Die Entscheidungen der künstlichen Intelligenz werden von der Aktorik verarbeitet und in konkrete Änderungen der Richtung oder des Geschwindigkeit des Fahrzeugs umgesetzt. Eine zentrale Aufgabe ist das Halten des Fahrzeugs auf der vorgesehenenen Spur. Eine weitere wichtige Aufgabe erfüllt die Aktorik mit dem Auslesen und Verteilen der Eigenposition des Fahrzeugs an alle Module.

Fahrspurerkennung

Dieses Modul liest Bilder aus Kameras aus und sucht nach Fahrspuren. Hier wird auf den Bildern zunächst eine Featureextraktion durchgeführt und Verzerrungen werden herausgerechnet. Ferner läuft in diesem Module eine erweiterte Version des in Abschnitt 3 beschrieben Trackingalgorithmus für Fahrspuren.

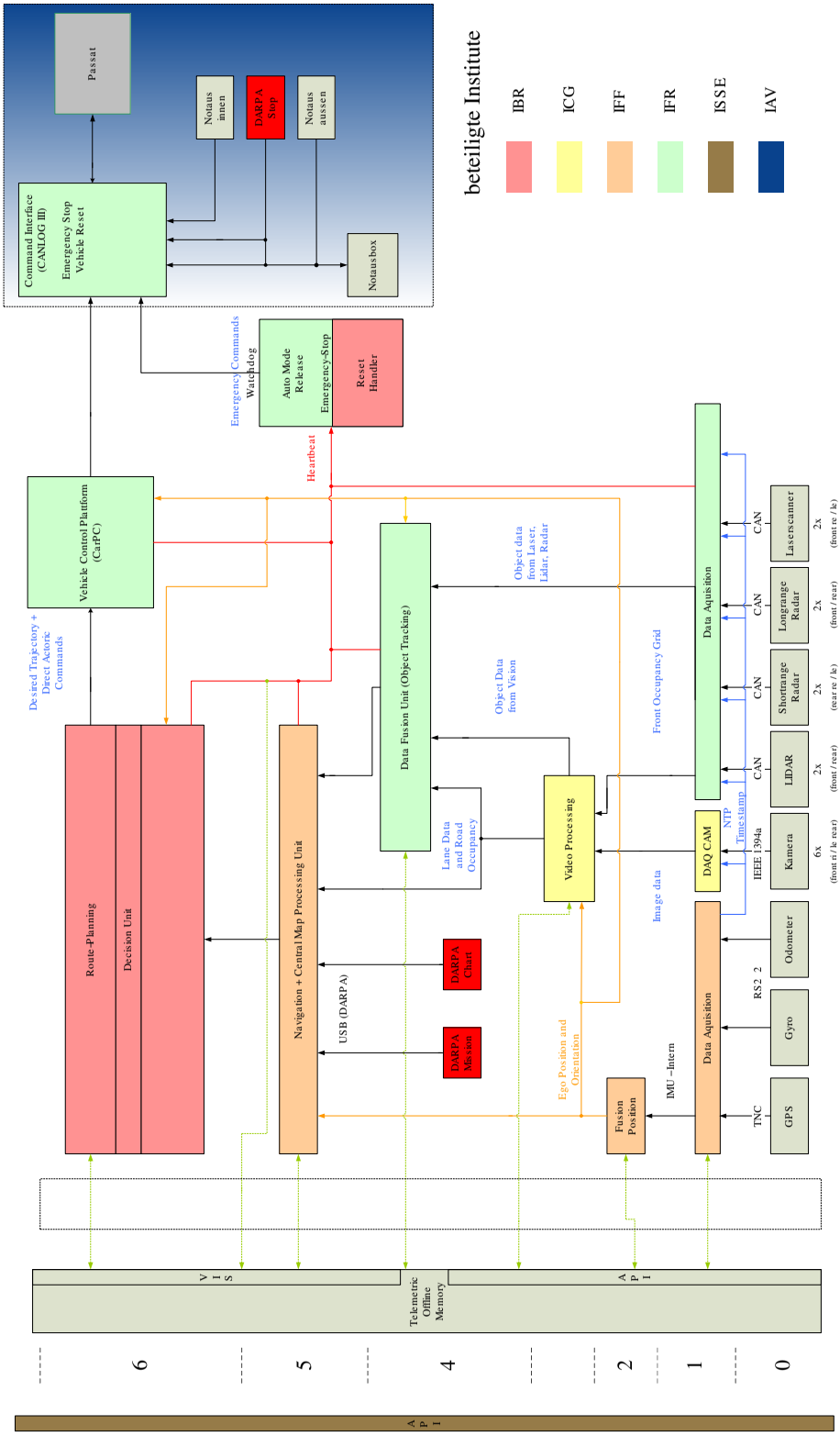


Abbildung 5.1: Architektur des CarOLO Projekts[a]

[a]CarOLO Projekt

Watchdog

Der Watchdog besitzt die Aufgabe die Verfügbarkeit von Modulen und Hardware zu überwachen. Dazu empfängt er in regelmäßigen Abständen Signale von jeder Applikation. Bleiben diese aus, so ist er in der Lage, Maßnahmen, von Neustart einer einzelnen Applikation bis hin zu dem Neustart eines ganzen Rechners, zu veranlassen.

5.2 Einbindung der Fusion

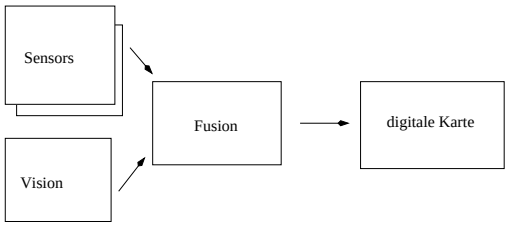


Abbildung 5.2: Einbindung der Fusion in das Gesamtprojekt

Entsprechend des Abschnitts 5.1 stellt die Fusion einen Filter im Pipes and Filters Muster dar. Sie wird durch drei Pipes von den Modulen Sensors und Vision gespeist. Abbildung 5.2 stellt dieses dar.

Eine Pipe entspringt dem Vision Modul und verschickt LaneNetworkObject-Objekte. Diese beschreiben eine Fahrspur durch eine Menge von Punkte auf der linken und rechten Fahrspur. Sie sind das Ergebnis des unter Abschnitt 3 beschriebenen Trackings von Fahrspuren.

Die zwei weiteren eingehenden Pipes kommen von zwei Instanzen des Moduls Sensors. Diese lesen die Sensoren aus und verschickt die so erhaltenen Daten über SensorSweep Objekte an die Fusion. Dabei enthält ein SensorSweep immer einen vollständigen Satz von SensorObjects, die zu einem Zeitpunkt in einem Sensor entstanden sind. Ferner kann er Daten über den Zustand eines Sensors (z.B. Verschmutzung oder Ausfall) übermitteln. Ein SensorObject enthält die in Abschnitt 2.3 beschriebenen Daten. Je nach Sensortyp können diese differieren. Das SensorObject ist in der Lage, alle Daten auf einheitliche Weise zu speichern. Dies macht die spätere Verarbeitung einfacher, da auf diese Weise Funktionalität nicht mehrfach implementiert werden muss.

Neben den eingehenden Pipes besitzt die Fusion eine ausgehende Pipe, die die Schnittstelle zur digitalen Karte darstellt. Auf diesem Weg werden zwei unterschiedliche Nachrichten

verschickt: Nachrichten über die Aktualisierung von Tracks und Nachrichten über die Terminierung von Tracks. Auf Nachrichten zur Erstellung von Tracks wurde verzichtet, da das erste Erscheinen einer Aktualisierungsnachricht einer Erstellungsnachricht gleichkommt. Die hier übermittelten Objekte sind vom Typ FusionObject. Dieser enthält die Zustandsgrößen des Kalmanfilters, die das Objekt beschreibenden Konturpunkte sowie Informationen über die Art und Güte des Objekts.

5.3 Architektur der Sensor-Daten-Fusion

Die Fusion verwendet ebenfalls das in [BMR+96] beschrieben Pipes and Filters Muster. Dieses bietet sich an, da es die Gesamtarchitektur des Projekts konsequent fortführt. Ferner ist die Hauptaufgabe der Fusion die Verarbeitung und Produktion eines Datenstroms. Auch dieses wird von dem Muster hervorragend gelöst. Ein weiterer Vorteil dieses Konzepts ist seine hohe Flexibilität. Es lassen sich mit wenig Aufwand neue Datenpfade einführen, ändern oder neue Filter zwischen zwei bestehende schalten. Auch ist die Möglichkeit zur parallelen Bearbeitung durch die Unabhängigkeit der einzelnen Filter gegeben.

Wie in Abbildung 5.3 dargestellt, existiert ein Hauptdatenflusspfad innerhalb der Software. In der Regel werden die im SensorObjectProcessor empfangenen SensorSweeps über das SensorObjectMapping zum KalmanFilterUpdate und schließlich zum FusionObjectDistributer geschickt. Der Filter PreTracking stellt einen alternativen Pfad dar, der der Erkennung von Sensorgeistern dient. Er erhält alle bisher nicht bekannten Objekte.

Als ein weiteres zentrales Designmuster wurde das Blackboard Muster [BMR+96] eingesetzt. Dieses beschreibt die Zusammenarbeit von mehreren (unabhängigen) Spezialisten durch einen zentralen Datenspeicher (Blackboard). Diese Zusammenarbeit wird durch eine zentrale Einheit koordiniert. In dieser Software stellt das SensorTrackStorage den zentralen Datenspeicher dar. Sie enthält alle Daten über die vorhandenen Tracks, die von den in Abbildung 5.3 beschriebenen Filter erzeugt, bearbeitet und verändert werden. Die Filter stellen die Spezialisten dar. Jede der einzelnen Stufen ist in der Lage, einen sehr begrenzten Auftrag zu erledigen und die Ergebnisse einerseits über eine Pipe an die nächste Stufe zu verschicken und andererseits die Daten im Blackboard zu aktualisieren. In diesem Muster spiegeln sich auch weitere, nicht in das Pipes and Filters Muster eingebundene, Stufen wieder. So ist hier der TrackCollector eingebunden, der die Aufgabe hat Tracks, zu untersuchen und gegebenenfalls aus dem Blackboard zu entfernen. Die Kontrolleinheit des Musters findet sich in der durch das Pipes and Filter Muster vorgebenen Datenflussstruktur wieder. Sie gibt vor, mit

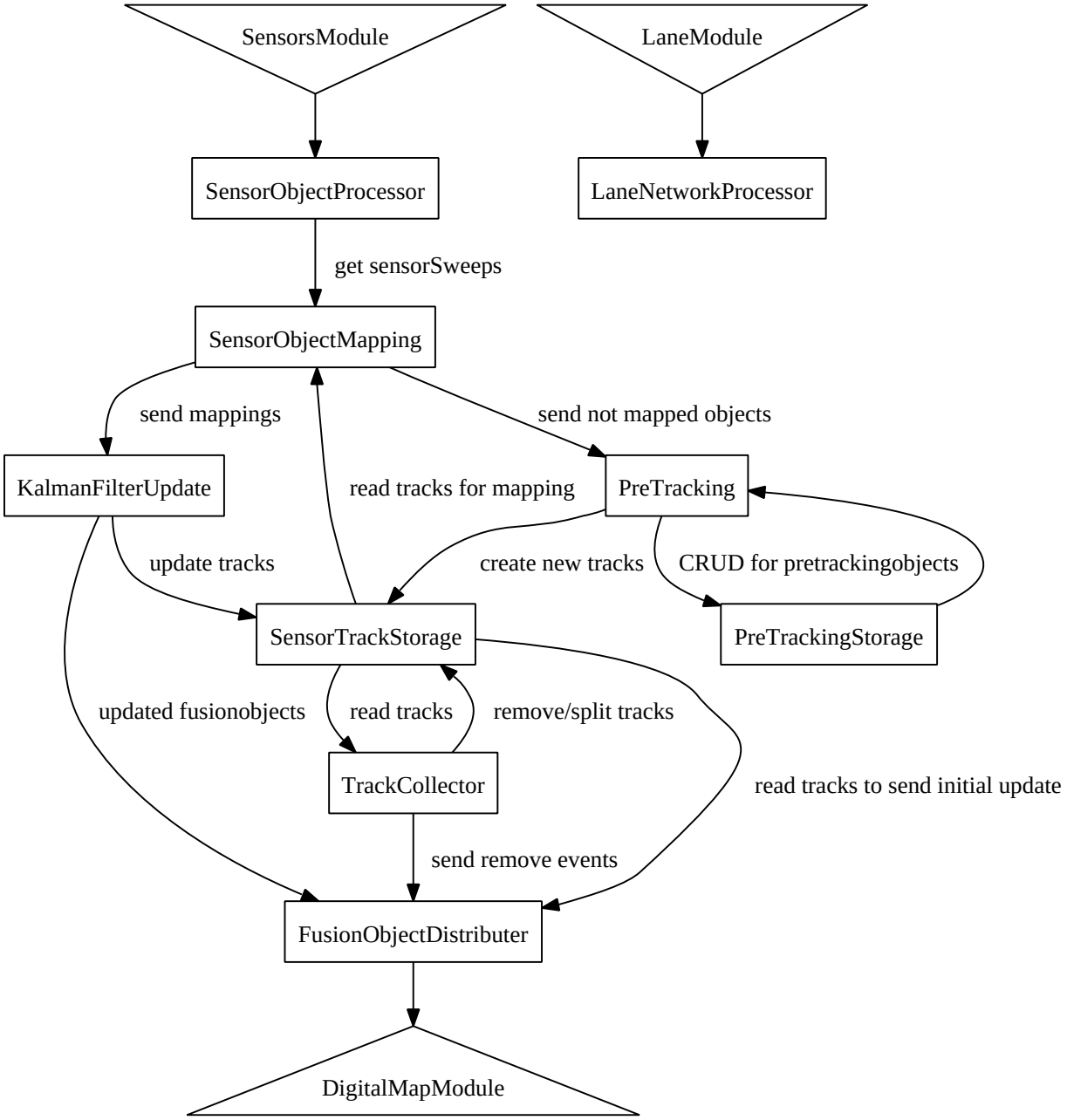


Abbildung 5.3: Pipeline Architektur der Fusion

welchen Daten des Blackboards sich die Spezialisten gerade beschäftigen sollen und wie die Lösung des Problems am besten zu erreichen ist.

5.3.1 FusionScheduler

Die von dem Pipes and Filters Muster ermöglichte Parallelität kann in diesem Projekt nur bedingt genutzt werden. Während der Entwicklung hat sich gezeigt, dass sich das Scheduling der für die einzelnen Stufen verwendeten Threads nicht eignet. Es traten wiederholt Verzögerungen von Daten in den Pipes auf, obwohl die Abnehmerfilter keine Aufgaben zu verrichten hatten. Aus diesem Grund wurde ein eigenes Scheduling entwickelt, das sich zum Ziel gesetzt hat, die Anzahl an Objekten in den Pipes zu minimieren.

Zu diesem Zweck wurden die bisher parallelen Stufen in einen einzelnen Thread überführt. Er überwacht die Füllung der Pipes und ruft entsprechend des Diagramms in Abbildung 5.3 die Funktionen auf, um die Objekte der Pipe abzuarbeiten. Es hat sich gezeigt, dass das durch das Betriebssystem zur Verfügung gestellte Scheduling für diese Anwendung nicht deterministisch genug ist. Dieses Verfahren stellt für diese Anwendung das Optimum dar, das den Durchsatz an Daten maximiert. Abbildung 5.4 stellt die beiden Varianten gegenüber. Zu sehen ist, dass in der durch threadsgesteuerten Variante die PreTrackingstufe nicht in der Lage ist, die Objekte zu verarbeiten und sich so ein Rückstau bildet.

5.3.2 SensorObjectProcessor

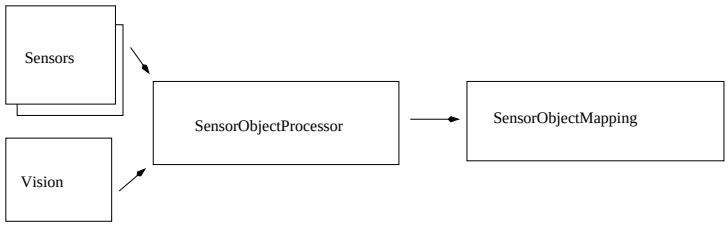


Abbildung 5.5: Datenfluss um den SensorObjectProcessor

Der SensorObjectProcessor ist die zentrale Stelle, die SensorSweeps entgegennimmt. Wie in Abschnitt 5.2 beschrieben, stellt ein SensorSweep eine Menge von Sensorobjekten dar, welche von der Fusion verarbeitet werden sollen. Diese werden von dem Sensormodul an die Fusion verschickt (siehe Abbildung 5.5). Die Übertragung der Daten geschieht über das TCP Protokoll, für die der SensorObjectProcessor einen Server implementiert. Die Daten werden

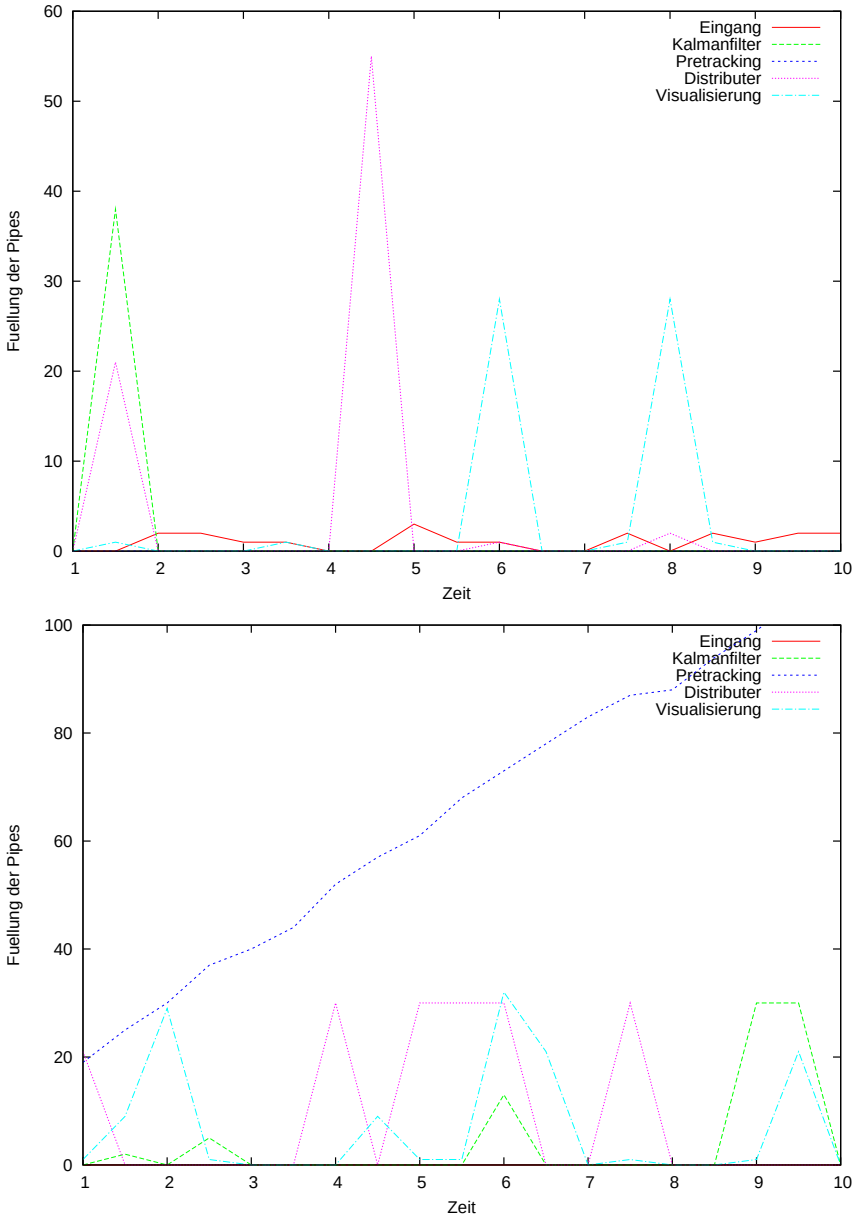


Abbildung 5.4: Vergleich der Füllstände der Pipes bei ca. 850 eingehenden Objekten pro Sekunde zwischen der durch den Scheduler (oben) und der durch Standardthreads (unten) gesteuerten Variante

bei Erreichen des Moduls ausgelesen, deserialisiert und dann über eine Queue an die SensorObjectMapping Stufe weitergeleitet. Eine weitere Aufgabe besteht darin, die eingehenden Sweeps an die Visualisierung zu verschicken.

5.3.3 LaneNetworkObjectProcessor

Abbildung 5.6: Datenfluss um den LaneNetworkObjectProcessor

LaneNetworkObjects beschreiben, wie in Abschnitt 5.2 dargestellt, Fahrspuren, die von dem Vision Modul erkannt wurden. Diese haben die Hauptaufgabe, das Fahrzeug von der KI auf der Fahrspur zu halten. Die Fusion empfängt diese Objekte ebenfalls, um eine bessere Typisierung von SensorTracks vornehmen zu können. Der LaneNetworkObjectProcessor implementiert einen TCP Server, der die Objekte empfängt und deserialisiert. Ferner bietet er die Möglichkeit, anderen Stufen der Software die Daten zur Verfügung zu stellen.

5.3.4 SensorObjectMapping

Das SensorObjectMapping Modul hat die Aufgabe, jedem SensorObject einen SensorTrack zuzuordnen bzw. zu entscheiden, dass kein für dieses Objekt gültiger Track existiert. Zu diesem Zweck wird das in Abschnitt 4.2 beschriebene Verfahren zum Zusammenführen von mehreren Objekten zu einem Track genutzt. So wird zu einem Objekt der optimale Track bestimmt.

Das Modul iteriert für jedes Objekt über alle Tracks und bestimmt die auftretenden Kosten einer Zuordnung. Die minimalen Kosten bestimmen die beste Zuordnung und das Objekt-Track-Paar wird an das KalmanFilterUpdate Modul übermittelt. Wird keine Zuordnung mit vertretbaren Kosten ermittelt, so wird das Objekt an das PreTracking Modul übermittelt, um ggf. ein neuer Track werden zu können.

Mit diesem Verfahren lassen sich sowohl unbekannte Objekte erkennen als auch bekannte Objekte mit Updates versorgen. Ferner ist es möglich, einem Track mehrere Objekte zuzuordnen, da keine feste Zuordnung besteht, sondern für jedes Objekt neu nach der optimalen

Zuordnung in allen Tracks gesucht wird. Auch hat eine kurzzeitige Fehl- oder Nichtzuordnung eines Objekts keine langfristigen Auswirkungen, da das nächste Update wieder korrekte Daten enthalten kann.

5.3.5 PreTracking

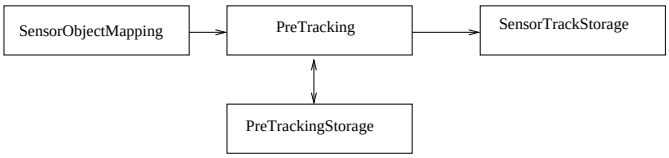


Abbildung 5.7: Datenfluss um das PreTracking

Im PreTracking Modul wird der Algorithmus aus Abschnitt 4.1 umgesetzt. Dieser beschreibt ein Verfahren zur Reduktion von Messrauschen und zur Verbesserung der Anfangswerte des Trackings. Mit ihm lassen sich Daten (z.B. Geschwindigkeiten) berechnen, die nicht von den Sensoren gemessen werden können.

Dieses Modul stellt einen gewissen Bruch mit dem Pipes and Filters Muster dar. Es empfängt zwar, wie in Abbildung 5.7 gezeigt, Objekte von dem SensorObjectMapping Filter schickt diese aber nicht weiter. Nach Abschluss des Algorithmus wird jedoch ein Track erzeugt und in das SensorTrackStorage gespeichert. Dies führt dazu, dass entsprechende Objektdaten im SensorObjectMapping Modul zugeordnet werden können und so nicht mehr an das PreTracking Modul, sondern an das KalmanFilterUpdate Modul geleitet werden.

Das PreTracking Modul enthält eine eigene Instanz des Storage Moduls, um Objekte vom Typ PreTrack zu speichern. Diese stellen potenzielle Tracks dar. Sie werden mit einem einfacheren als dem 6D-Kalmanfilter verfolgt. Dieser hat, die Aufgabe Objekte zu verfolgen und zusätzlich Initialisierungsdaten für einen späteren Track zu erzeugen, da nicht alle Sensoren alle benötigten Daten liefern.

Der hier verwendete Kalmanfilter besitzt folgendes Systemmodell:

$$\hat{x}_k = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix}$$

$$P_k = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dieser Filter ist ausreichend, um Objekte verfolgen zu können und dennoch nicht die Rechenleistung zu stark zu beanspruchen.

Die im PreTracking verwendeten Objekte (PreTracks) enthalten neben den Zustandsdaten für den Kalmanfilter einen Zähler, der die Wiedererkennungen speichert. Wird ein PreTrack in einem SensorSweep nicht wiedererkannt, so wird dieser dekrementiert beziehungsweise inkrementiert. Um ein SensorObject einem PreTrack zuzuordnen, wird eine Liste mitgeführt, die einem Sensortyp eine Sensor ID zugeordnet. Auf diese Weise läßt sich ein PreTrack aus der Datenbank extrahieren. Hierbei wird ein HashIndex verwendet. Um nicht einen HashIndex für jeden Sensortyp mitführen zu müssen, wird in dem HashIndex ein gemapter Wert, bestehend aus Sensortyp und Sensor ID, gespeichert. Dieser errechnet sich zu $SensorType * 1000 + SensorID$. Dieser Formel liegt die Annahme zugrunde, dass die Sensor ID stets kleiner ist als 1000, was durch die Spezifikation der Sensoren sichergestellt ist. Dieses Mapping macht die Software flexibel gegenüber der Integration neuer Sensoren, da lediglich an einer Stelle ein neuer Sensortyp definiert werden muss.

Wird ein PreTrack hinreichend verifiziert, wird er schließlich aus der PreTrackStorage Instanz entfernt und den regulären Tracks hinzugefügt. Dazu werden die bereits getrackten Daten übernommen. Ein so erzeugter Track wird der KI noch nicht übermittelt. Diese erhält die Daten erst nach einer weiteren Verifizierung durch das SensorObjectMapping. Daraus entsteht der Vorteil, dass der Wiedererkennungspfad komplett verifiziert wurde. Zusätzlich sind nach dem ersten Update durch den Kalmanfilter bereits korrekte Varianzen und nicht nur Initialwerte vorhanden.

5.3.6 KalmanFilterUpdate

Abbildung 5.8: Datenfluss um das KalmanFilterUpdate

Sensorobjekte, die durch das SensorObjectMapping einem Track zugeordnet wurden, werden an das KalmanFilterUpdate Modul versandt (siehe Abbildung 5.8). Dieses Modul hat

die Aufgabe, die Konturpunktzuordnung durchzuführen und die Daten an den für diesen Sensortyp richtigen Kalmanfilter zu übergeben. Die Kalmanfilter unterscheiden sich durch die eingehenden Messdaten (siehe Abschnitt 2.3). Für jeden Sensortyp ist so eine andere Messmatrix H_k nötig, um die Werte auf die vom Filter in der Statematrize vorhandenen umrechnen zu können. Weiter lassen sich auf diese Weise die Rauschmatrizen unterschiedlich einstellen. Zunächst wird der in Abschnitt 4.3.1 beschriebene Algorithmus zur Zuordnung von Track- zu Sensorobjektkonturpunkten durchgeführt. Bei diesem Algorithmus wird eine möglichst optimale Zuordnung bezüglich des Abstandes zweier Punkte gesucht. Die so errechnete Zuordnung wird anschließend an die in Abschnitt 4.3.2 beschriebene Aktualisierung der Trackdaten übergeben. In das SensorTrackStorage wird der gefilterte Track aktualisiert und anschließend an den FusionObjectDistributer übermittelt. Anschließend sorgt dieser für die Übermittlung an die KI.

5.3.7 TrackCollector

Abbildung 5.9: Datenfluss um den TrackCollector

Die in Abschnitt 4.5 (Terminierung), 4.2 (zeitgesteuertes Verschmelzen) und 4.4 (Aufspaltung) beschriebenen Algorithmen sind in diesem Modul implementiert. Der TrackCollector sorgt für das Aufspalten, Verschmelzen sowie das Terminieren von Tracks. Dieser Vorgang wird mit einem Takt von 500 ms durchgeführt. Um eine schnelle Reaktion auf aufzuspaltende/verschmelzende Objekte zu realisieren und dennoch nur eine geringe Rechenlast zu erzeugen, ist dies ausreichend. Zunächst werden in der Trackdatenbank die Daten gefunden, die mehr als eine Sekunde nicht aktualisiert worden sind und sie werden entfernt. Im Anschluss wird zunächst das Trackmerging Verfahren angewandt, um Tracks mit anderen Track zu verschmelzen. Darauf folgend kommt der Tracksplitting Algorithmus zum Einsatz. Er wird auf jeden noch in der Datenbank vorhanden Track angewandt, um ihn gegebenenfalls aufzuspalten.

Wie in Abbildung 5.9 beschrieben, sendet der TrackCollector DeleteEvents für jeden gelöschten Track an den FusionObjectDistributer. Für gespaltetene Tracks werden ebenfalls DeleteEvents geschickt. Die im gleichen Zuge neu eingefügten Tracks erzeugen jedoch keine UpdateEvents, um sie vor dem Bekanntmachen der KI zunächst durch die Sensoren bestätigen zu lassen.

5.3.8 FusionObjectDistributer

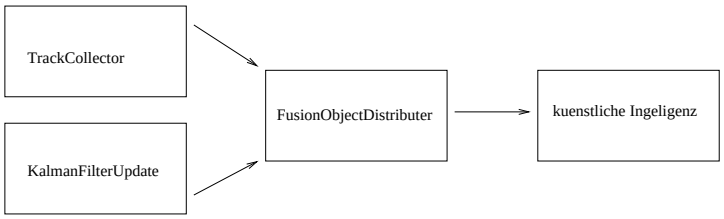


Abbildung 5.10: Datenfluss um den FusionObjectDistributer

Das Modul FusionObjectDistributer besitzt die Aufgabe, die von dem Modul KalmanFilterUpdate erzeugten Updateinformationen für Tracks sowie die vom Modul TrackCollector erzeugten Informationen über das Terminieren von Tracks an die künstliche Intelligenz zu übermitteln. Abbildung 5.10 beschreibt diesen Zusammenhang.

Hierzu wird eine TCP-Verbindung zur KI aufgebaut. Anschließend wird der gesamte Trackbestand verschickt, um einen gemeinsamen Zustand zwischen KI und Fusion herzustellen. Im Folgenden werden dann Update- und Delete-Events an die KI übersandt. Dies besitzt den Vorteil gegenüber dem Verschicken des Gesamtbestandes, dass nur ein Bruchteil der Datenrate benötigt wird. Bricht die Verbindung ab, so wird erneut versucht, sie wieder herzustellen. Eine weitere Aufgabe des Moduls ist es, das Versenden der erzeugten Daten an die Visualisierung zu realisieren.

5.3.9 Storage

Das Storage Modul ist die zentrale Datenspeichereinheit der Software. Sie ist in der Lage, Objekte abzuspeichern und diese zu indizieren, um bei Bedarf effizient darauf zugreifen zu können. Außerdem wird das Modul dazu genutzt, um Objekte länger speichern zu können. Es stellt den autoritativen Ort für Objekte innerhalb der Fusion dar. Insofern speichert jeder Filter das Ergebnis seiner Transformation am Ende in diesem Speicher, so dass ein anderer Filter die Daten weiter bearbeiten kann.

Anforderungen an das Storage Modul

Die Fusion muss in der Lage sein, alle eingehenden Daten zu verarbeiten, ohne dass sich ein Rückstau bildet. Das Storage muss somit mehr als 10000 Lese-/Schreibzugriffe pro Sekunde

(unter der Annahme, dass 100 Objekte pro 100 ms durch alle fünf Stufen laufen und jeder nur eine Schreib- und Leseoperation durchführt) bearbeiten.

Das Storage muss keine Daten physikalisch dauerhaft speichern, da die Sensortracks nur eine sehr begrenzte Lebensdauer haben und problemlos nach einen Neustart der Software wieder erzeugt werden können. Um die hohe Performance der Fusion erreichen zu können, muss das Storage effiziente Zugriffsmöglichkeiten auf die gespeicherten Objekte bieten. Da die einzelnen Stufen teilweise parallel arbeiten, muss threadsafty gewährleistet sein und immer eine konsistente Sicht auf die Daten geboten werden.

Umsetzung der Anforderungen

Für das Storage wird eine Standardarchitektur für eine relationale Datenbank gewählt (siehe Abbildung 5.11). Sie besteht aus drei Schichten, die miteinander kommunizieren und die jeweils untere Schicht kapseln. Parallel zu diesen Schichten existiert ein Metadatenteil, der die gespeicherten Daten näher beschreibt.

Eine Instanz der Storage Klasse (siehe Abbildung 5.12) wird durch ein Trait-Template [DNS+06] beschrieben. Traits bieten die Möglichkeit, eine objektorientierte Struktur über die bestehenden Möglichkeiten, wie Ableitungen und Assoziationen, hinaus anzupassen. Sie vereinfachen die Wiederverwendbarkeit von Softwarekomponenten. Dies geschieht, indem eine Menge von Funktionen zur Compiletime austauschbar werden. Auf diese Weise läßt sich das Gesamtverhalten einer Klasse oder eines Moduls ändern. Im Fall der Storage enthält das Trait-Template alle Metadaten, die nötig sind, um einen Objekttyp zu speichern. Eine Instanz kann immer nur einen Objekttyp aufnehmen. Die Metadaten setzen sich aus dem Typ und einer Funktion zusammen, die aus einem Objekt einen eindeutigen Integerschlüssel erzeugt. Mit diesem Schlüssel werden später alle Operationen, wie z.B Update oder Delete, ausgeführt.

Die eigentliche Datenspeicherung der Speichersystemschicht wird von dem Objekt StorageBlock realisiert. Ein StorageBlock hat eine gewisse Menge von Speicherplätzen, die Objekte aufnehmen können. Er hält Speicherplatz vor, der aber nicht zwangsläufig genutzt werden muss. Dies hat den Vorteil, dass langsame Speicherallokationen zur Laufzeit weitestgehend unterbleiben und nur bei Bedarf und dann blockweise durchgeführt werden müssen. Um freie Speicherplätze in einem Block zu finden, bietet dieses Objekt neben den Operationen „store" und „get" noch die Möglichkeit, nach der nächsten freien Speicherposition zu suchen.

Das Storage Objekt realisiert zunächst die zweite Schicht, das Zugriffsystem mittels Tupel Identifer (TID)[psq06]. Der TID wird von der Hilfsklasse StorageIndex realisiert. Der

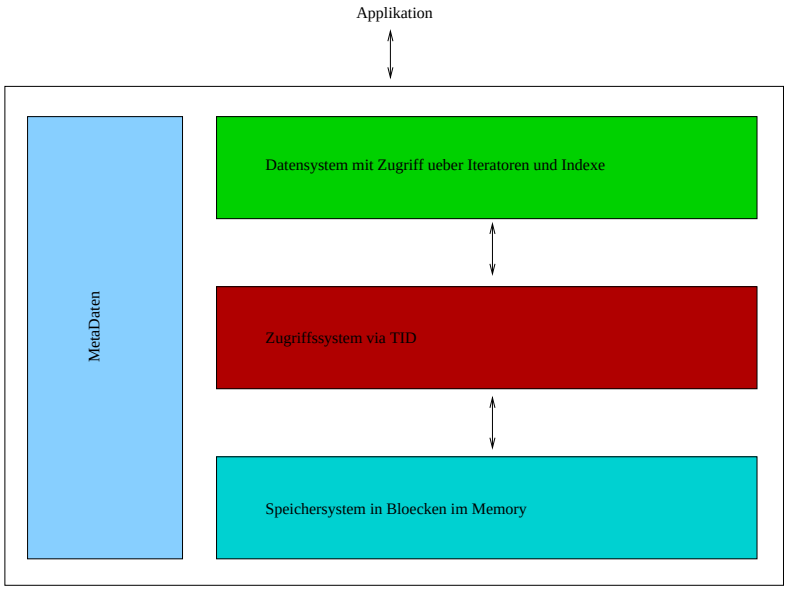


Abbildung 5.11: Standardarchitektur einer Datenbank ohne Transaktionsverwaltung

StorageIndex identifiziert exakt einen Speicherplatz (Zelle) in einer Storage, bestehend aus StorageBlocknummer und StorageBlockspeicherplatz. Der Vorteil von TIDs ist die Entkoppelung von Daten und Speicheradresse. Eine Anforderung war, dass ein effizienter Zugriff und zu jeden Zeitpunkt eine konsistente Darstellung zu realisieren ist (siehe Abbildung 5.13). Mittels des TID Konzepts wird nur an einer Stelle das Objekt gespeichert. Indizes z.B. speichern nur die TID ohne, selbst eine Kopie des Objekts vorzuhalten. Auf diese Weise werden beide Forderungen erfüllt.

Die dritte Schicht realisiert den Zugriff der Applikation auf die Datenbank. Dies wird einerseits von dem Storage Objekt und andererseits von Indexklassen realisiert.

Die Storage Klasse realisiert die Schreiboperationen und den Zugriff auf die gesamt Datenbank. Zur Manipulation der Daten steht ein Create, Read, Update, Delete (CRUD) Interface zur Verfügung [Wik07a]. Diese Operationen werden durch POSIX-RWLocks [pos04] abgesichert. RWLocks haben den Vorteil, dass sie Multiple-Read-Single-Write erlauben. Dies verbessert die Read Performance der Storage und somit die Forderung nach einem hohen Durchsatz. Um Objekte zu aktualisieren, gibt es zwei Möglichkeiten. Bei der einen wird eine Kopie des Objekts hereingereicht und dann in die Datenbank geschrieben. Dies kann zu Problemen führen, wenn mehrere Threads parallel auf Kopien eines Objekts arbeiten. Abbildung 5.14 zeigt den Fall, dass Thread A und Thread B sich für Arbeiten mit demselben Objekt eine Kopie desselben holen. Beide aktualisieren Felder des Objekts und speichern es dann zurück in die Datenbank. Thread A speichert als erstes. Da Thread B seine Kopie

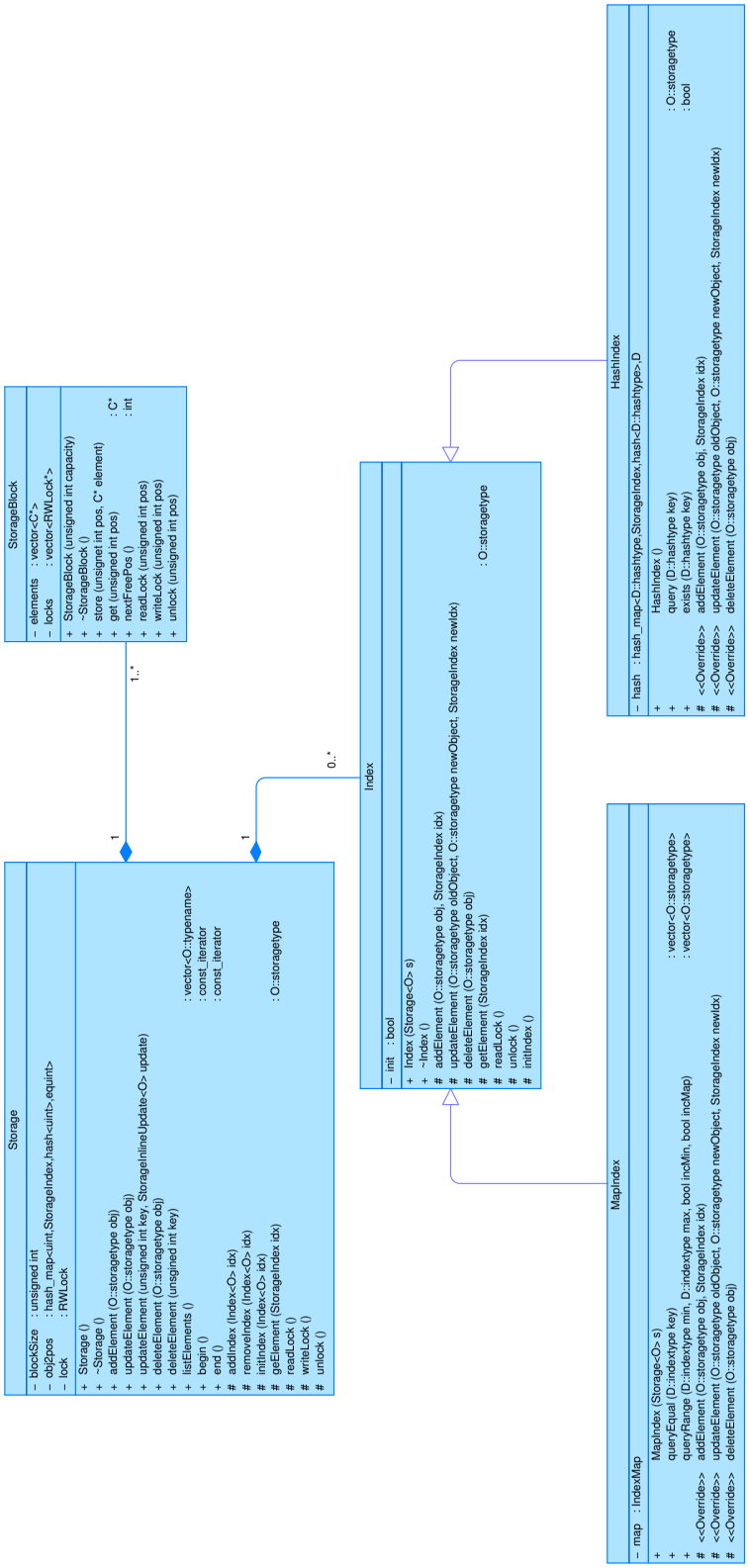


Abbildung 5.12: Klassendiagramm des Storage Moduls

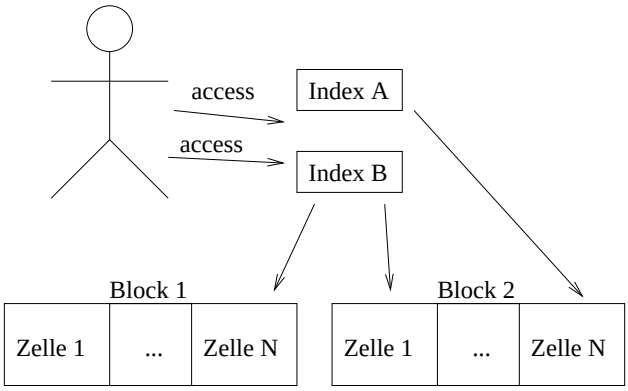


Abbildung 5.13: Speicherung der Daten mittels TID und Zugriff via Indizes

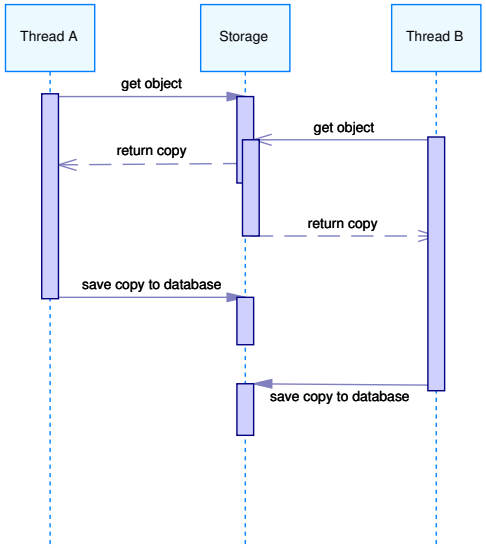


Abbildung 5.14: Beispiel für fehlerhaftes Update

erst später speichert, gehen die Änderungen von Thread A verloren. Um dieses Problem anzugehen, bietet das Storage eine zweite Möglichkeit ein Objekt zu aktualisieren. Es ist möglich, dem Storage eine Funktion zu übergeben, die das in der Datenbank gespeicherte Objekt als Parameter übergeben bekommt. Die Funktion kann einzelne Felder aktualisieren und die restlichen unangetastet lassen. Ein weiterer Vorteil dieser Methode ist, dass die übergebene Funktion innerhalb des WriteLocks ausgeführt wird. Es ist damit sichergestellt, dass niemand eine Kopie des Objekts während des Updates erhalten kann. Zum Zugriff auf alle Elemente des Storage implementiert die Storage Klasse zwei Methoden. Die erste macht eine Kopie aller Elemente und reicht diese an die Applikation weiter. Dies ist allerdings sehr ineffizient, da Copy Operationen meist langsam sind. Eine weit effizientere Methode sind Iteratoren auf dem Datenbestand [Ste05]. Sie machen keine Kopie des Objekts, sondern reichen

direkt Referenzen auf das Objekt im StorageBlock. Eine Veränderung ist nicht möglich, da sonst sehr viel Logik nötig wäre, um die Indizes aktuell zu halten. Um die Thread-Sicherheit zu gewährleisten, sind Rowlevel RWLocks im StorageBlock vorhanden. Der Iterator sperrt entsprechend seiner Funktion das jeweilige Objekt im ReadMode. Dies verhindert, dass das Objekt verändert oder gelöscht wird.

Um auf einzelne Objekte eines Storage zuzugreifen, sind Indizes realisiert worden. Indizes sind Zugriffspfade für direkten oder assoziativen Zugriff. An sie können Anfragen gestellt werden, um effizient ein Ergebnis, das den Parametern der Anfrage entspricht, zu erstellen. In diesem Modul wurden Hash- und MapIndizes auf Basis der STL [Inc06] realisiert. Die Definition eines Indizes wird wie bei der Storage über Traits realisiert.

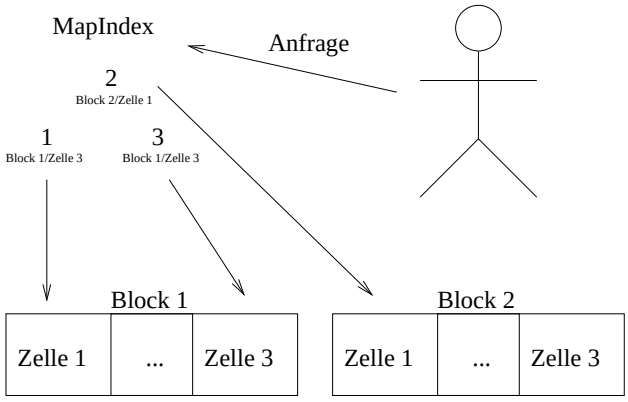


Abbildung 5.15: MapIndex

Der MapIndex bietet die Möglichkeit mit einer Komplexität von $O(n * \log n)$ auf Einzeldatensätze zuzugreifen. Dieses wird durch die Anordnung von Schlüsselwerten als Baumstruktur realisiert (siehe Abbildung 5.15), wobei jeder Schlüssel eine Menge von Speicheradressen von Objekten hält, die diesem Schlüssen entsprechen. Im Vergleich zum HashIndex ist er in der Lage, unter einem Schlüssel mehrere Datensätze abzulegen. Ein weiterer Vorteil ist die Unterstützung von Range Querys ($\leq, \geq, >, <$). Diese Anfragen haben stets die gleiche Komplexität [Inc06]. Der Trait für einen MapIndex enthält nur den zu indizierenden Datentyp sowie eine Funktion, die einem Objekt einen Schlüssel zuordnet.

Der HashIndex bietet die Möglichkeit mit der Komplexität von $O(1)$ auf einzelne Datensätze zuzugreifen. Der HashIndex ist viel effizienter als der MapIndex, allerdings lassen sich nur Objekte indizieren, die einen eindeutigen Schlüssel besitzen. Sie werden mit Hilfe einer Hashdatenstruktur gespeichert. Diese enthält neben einem Schlüsselwert die genaue Speicherposition des Objekts (siehe Abbildung 5.16). Eine Besonderheit an der Realisierung dieses Indexes ist es, dass ein Objekt die Anzahl der Schlüssel im Bereich von $0 \ldots n, n \in N$ frei

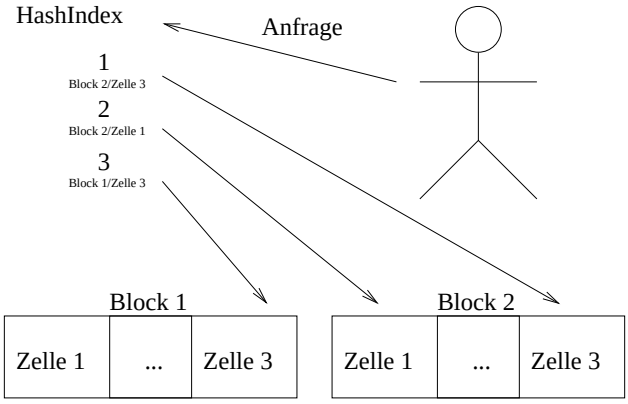


Abbildung 5.16: HashIndex

wählen kann. Dies ist nötig, um einem PreTrack mehrere Sensor IDs zuordnen zu können (siehe Abschnitt 5.3.5). Ein Trait für einen HashIndex enthält den zu indizierenden Datentyp, einen Wert, dem die Bedeutung NULL (nicht indizieren) zugewiesen wird, eine Equalfunktion, um zwei Elemente des zu indizierenden Datentyps zu vergleichen, sowie eine Funktion, die auf einem Objekt eine Menge von Schlüsseln erzeugt.

5.4 Visualisierung des Fahrzeugumfeldes

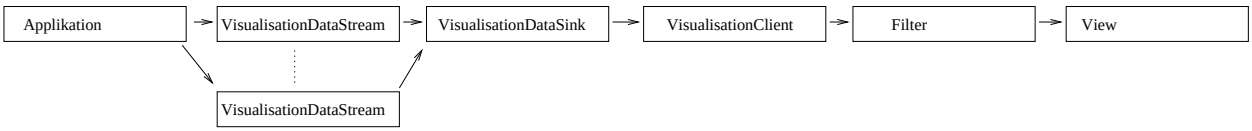


Abbildung 5.17: Datenfluss von der Applikation zur Visualisierung

Zur Beurteilung der Qualität und dem Erkennen von offensichtlichen Fehlern ist es von größter Wichtigkeit, die durch die Sensor-Daten-Fusion bearbeiteten Daten angemessen visualisieren zu können. Dies beinhaltet sowohl die Polygonzüge der getrackten Objekte als auch die Darstellung von Kerndaten wie Geschwindigkeit, Azimuth oder Varianzwerte.

Hierzu wurde im Rahmen des CarOLO Projekts eine Visualisierungssoftware entwickelt. Sie ist in der Lage, Daten der Sensor-Daten-Fusion sowie andere relevante Daten des Fahrzeugs darzustellen.

Das Konzept der Software basiert auf einer Teilung des Datenflusses. Abbildung 5.17 zeigt die an der Übermittlung beteiligten Module. Die Grundidee besteht darin, die Visualisierung

soweit wie möglich von der Applikation, die die Daten erzeugt, zu entkoppeln. Zu diesem Zweck instanziert jede Applikation ein Objekt des Typs VisualisationDataSink. Es übernimmt die Kommunikation mit einer oder mehreren Visulisationssoftwareinstanzen. Ferner erzeugt die Applikation eine oder mehrere Instanzen des Typs VisualisationDatastream. Ein Objekt dieser Klasse stellt einen Datenstrom für ein spezifisches Protokoll dar.

Um ein Datum von der Applikation an die Visualisierung zu schicken, wird dieses dem VisualisationDatastream übergeben. Er sorgt für die Überstellung an die VisualisationDataSink. Der Vorteil der Datastreams liegt darin, dass eine DataSink für unterschiedliche Daten genutzt werden kann. Die DataSink verteilt die Daten an verbundene Visualisierungen. Ist zu dem Zeitpunkt des Auftretens des Datums keine Software verbunden, so wird das Datum vernichtet. Durch das permanente Verschicken der Daten an die DataSink wird der Kontrollfluss der Applikation auch dann nicht wesentlich verändert, wenn keine oder sehr viele Visualisierungen mit der Applikation verbunden sind. Dies ist von Bedeutung, da auf diese Weise in der Leistungsfähigkeit einer Software kein messbarer Unterschied festzustellen ist. Bei der Anzahl der Visualisierungen hat sich vielmehr die Netzwerkinfrastruktur als ein Flaschenhals erwiesen. Durch die große Menge von Daten, die durch hohe Aktualisierungsraten erzeugt werden, kann es zu deutlichen Qualitätsverlusten innerhalb des im Fahrzeug verlegten Ethernets kommen.

Auf Seite der Visualisierungssoftware spaltet sich die Infrastruktur in drei Elemente. Der VisualisationClient stellt eine Abstraktion des Protokolls dar. Er baut die Verbindung zu der Applikation auf. Durch diese Invertierung der Datenflussrichtung sind keine Änderungen an der Konfiguration einer Applikation zur Anbindung einer Visualisierung nötig. Außerdem bietet der Client die Möglichkeit, Datenströme zu registrieren, die fortan an die Visualisierung verschickt werden. Die nächste Instanz im Datenfluss ist der Filter. Dieses Objekt wird für jeden Datenstrom speziell angepasst. Er hat die Aufgabe, die empfangenen Daten für die Darstellung aufzubereiten. Der View stellt die eigentliche Darstellungskomponente dar. Er erzeugt ein Fenster und stellt die durch den Filter aufbereiteten Daten dar. Der Vorteil der Trennung von Filter und View ist, dass View von mehr als einem Filter genutzt werden können. So wurde beispielsweise ein 2D-Graph entwickelt, der Kurvendiagramme darstellt. Dieser View wird von diversen Filtern genutzt, um Daten über der Zeit aufzutragen. Eine weitere wichtige Variante ist, dass mehrere Filter einen View nutzen. Dies ist nötig, um unterschiedliche Informationen gleichzeitig darstellen zu können. Die Visualisierung enthält beispielsweise einen EgoView, der sowohl das Umfeld des Fahrzeugs als auch die Entscheidungen und Planungen der künstlichen Intelligenz darstellt.

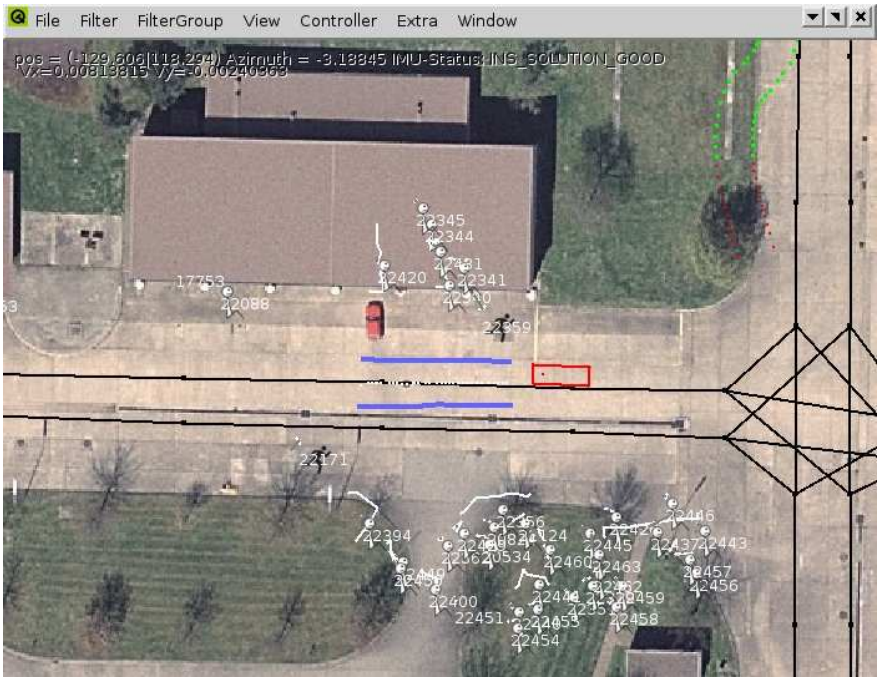


Abbildung 5.18: Visualisierung des Fahrzeugumfelds

Abbildung 5.18 stellt das Fahrzeugumfeld auf dem Testgelände des Versuchsträgers dar. Die hier gewählte Darstellung zeigt den EgoView, in dem die Daten der Sensor-Daten-Fusion eingeblendet sind. Als Datenformat wurde das FusionObject gewählt. Dieses bietet den Vorteil, dass es bereits erstellt wurde, um an die künstliche Intelligenz verschickt zu werden. Zur Übermittlung an die Visualisierung müssen die Daten lediglich kopiert werden. Dargestellt werden die Konturzüge der einzelnen Objekte. Vom Mittelpunkt abgehend, wird ein Bewegungsvektor gezeichnet. Durch Markieren eines Objekt mit der Maus werden weitere Daten eingeblendet. Hierbei werden in Textform dargestellt: Position in Weltkoordinaten, Position in Fahrzeugkoordinaten, geringster Abstand zum Fahrzeug, Geschwindigkeit, Beschleunigung und Kurswinkelbeschleunigung. Die Auswahl der Daten leitet sich hauptsächlich aus den Anforderungen der künstlichen Intelligenz ab. Die in Abschnitt 4.3.5 dargestellte Klassifikation in statische und dynamische Objekte wird durch kleine Piktogramme (sitzendes/laufendes Männchen) abgebildet. Ferner sind in der Darstellung die Straßenkarte (schwarze Linien) sowie die erkannte Fahrspur (blaue Linien vor dem Fahrzeug) dargestellt.

6 Messungen

Um die Güte der Software dieser Arbeit zu bestimmen, wurden Messungen durchgeführt. Sie sollen in diesem Abschnitt vorgestellt werden. Dazu ist es nötig, die genauen Positionen der durch die Sensoren erkannten Objekte zu bestimmen. Dies mit real existierenden Objekten durchzuführen, stellt einen erheblichen Aufwand dar, der im Rahmen dieser Arbeit nicht zu leisten war. Aus diesem Grund wurden alle bis auf eine Messung an simulierten Daten vorgenommen. Im folgenden Abschnitt wird beschrieben, wie diese Daten erzeugt wurden. Darauf folgend schließen sich Beschreibungen verschiedener Messdaten an.

6.1 Testdatenerzeugung

Für die Messungen der Güte der Sensor-Daten-Fusion werden Testobjekte benötigt. Diese müssen Objekte darstellen, wie sie in der Realität auftreten können.

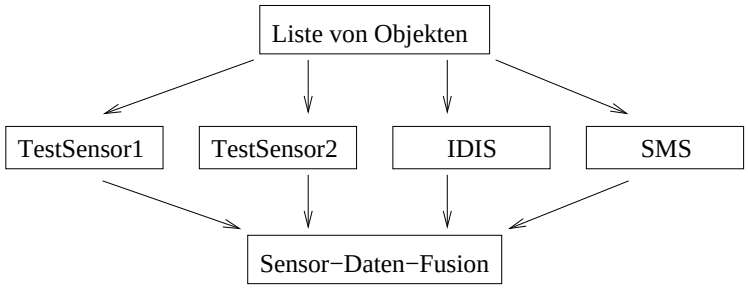


Abbildung 6.1: Komponenten der Simulation

Die Softwarekomponente, die zur Simulation eingesetzt wird, besteht aus zwei Subkomponenten (siehe Abbildung 6.1). Zunächst wird eine Liste aufgebaut, die eine gewisse Anzahl von Objekten enthält. Jedes dieser Objekte ist unabhängig von den anderen. Sie können sich mit unterschiedlichen Geschwindigkeiten bewegen und unterschiedliche Formen annehmen. Die Anzahl der Objekte ist frei wählbar. Auf diese Weise können Durchsatzmessungen durchgeführt werden.

Zusätzlich lassen sich unterschiedliche Bewegungsmodelle für die Objekte wählen. Objekte können sich linear mit konstanter Geschwindigkeit oder als Sinusschwingung fortbewegen. Die Sinusschwindung besitzt den Vorteil, dass ihre Bewegung nicht von dem 6D-Kalmanfilter der Sensor-Daten-Fusion abgebildet werden kann. Somit lassen sich damit Grenzfälle simulieren. Die simulierten Objekte können sich in einem Rechteck von 400x400 m um das still stehende Fahrzeug bewegen. Trifft ein Objekt auf den Rand, so wird es mit einer neuen ID neu initialisiert. Auf diese Weise ist sichergestellt, dass sich immer die gleiche Anzahl an Objekten im System befindet.

Die zweite Subkomponente stellt einzelne Sensoren dar. Diese Schicht legt das Rauschen und die tatsächlichen Konturen entsprechend der Eigenschaften eines Sensors fest. Für diese Simulation wurden vier Sensorsimulationen mit unterschiedlichen Konturmöglichkeiten und Sichtbereichen entwickelt. Sie greifen auf eine Objektliste zu, lesen die Daten aus und erzeugen daraus SensorSweeps, die an die Sensor-Daten-Fusion übermittelt werden. Dies alles geschieht zyklisch und parallel. Jede Sensorsimulation sowie die Objektliste besitzt ihren eigenen Takt. Sie arbeiten mit einem Takt von ca. 100ms, wobei ein gewisser Jitter als Rauschen von jeder Sensorsimulation erzeugt wird.

Zur Erzeugung von Zufallswerten wird der Zufallsgenerator der OpenCV Bibliothek eingesetzt. Er erzeugt normal verteiltes Rauschen und entspricht dem FIPS 186-2 / Cert 245 Standart [18600]. Jede der Softwarekomponenten bedient sich des gleichen Zufallsgenerators.

Die einzelnen Sensorsimulationen sind in der Lage, unterschiedliche Konturen zu erzeugen:

	Quadrat	Linie	l. Fahrzeug	r. Fahrzeug	Dreieck	Pentagram	Punkt
TestSensor1		√		√	√	√	√
TestSensor2	√	√	√		√	√	
IDIS		√					
SMS							√

Tabelle 6.1: Geometrische Formen der simulierten Sensoren

Durch die Zweiteilung der Simulation erreichen die Sensor-Daten-Fusion mehr Objekte als in der Simulation verwaltet werden. Dies liegt daran, dass unterschiedliche simulierte Sensoren ein Objekt verarbeiten und so mehrfach an die Fusion geschickt werden. So wird bespielsweise ein linienförmiges Objekt sowohl von TestSensor1, TestSensor2 als auch durch den IDIS Sensor verarbeitet. Auf diese Weise erreichen die Fusion drei SensorSweeps, die das gleiche Objekt enthalten.

6.2 Durchsatzmessung

Um das Verhalten der Software bei unterschiedlichen Lastsituationen zu messen, wurde mit der Simulation eine unterschiedliche Menge von Objekten erzeugt. Dabei wurde die Latenz gemessen, die ein Objekt von der Erzeugung durch einen simulierten Sensor bis zum Erzeugen des FusionObjects benötigt. Dabei wird das arithmetische Mittel aller Latenzen aufgetragen. Auf ein Rauschen wurde bei dieser Messung verzichtet. Die Objekte folgen dem linearen Bewegungsmodell.

Abbildung 6.2: Messung der Latenz der Sensor-Daten-Fusion

Abbildung 6.2 trägt die Anzahl der simulierten Objekte (nicht die tatsächlich die Fusion erreichenden Objekte) gegenüber der Latenz auf. Wie zu erkennen ist, folgt die Latenz einer Parabel. Die Latenz, welche in der Regel mit realen Daten am Versuchsträger auftritt, bewegt sich zwischen 30 und 40 ms.

6.3 Abweichungen von der Realität

Dieser Abschnitt zeigt Messungen der Abweichungen von Objekten gegenüber der Realität. Dabei wird auf die Simulation zurückgegriffen, da Positions- oder Geschwindigkeitsbestimmungen in der Realität nur sehr schwierig durchzuführen sind.

In den folgenden Diagrammen werden die Varianzen der Position und der Geschwindigkeit des Kalmanfilters auftragen, da sie ein Maß für die Abweichung darstellen. Sie beschreiben je ein Objekt, das sich mit unterschiedlichem Rauschen und unterschiedlichen Bewegungsmodellen fortbewegt.

Die simulierten Objekte stellen ein Punktobjekt dar. Dieses bewegt sich mit $10\frac{m}{s}$ entlang der y-Achse des innertialen kartesischem Koordinatensystems. Objekte mit linearem Bewegungsmodell behalten ihr x-Position bei. Objekte mit sinusförmigen Bewegungsmodell folgen einem Sinus mit 20m Amplitude sowie einer Phasendauer von 5 Sekunden. Das Rauschen der Messwerte beträgt 0m oder 1m.

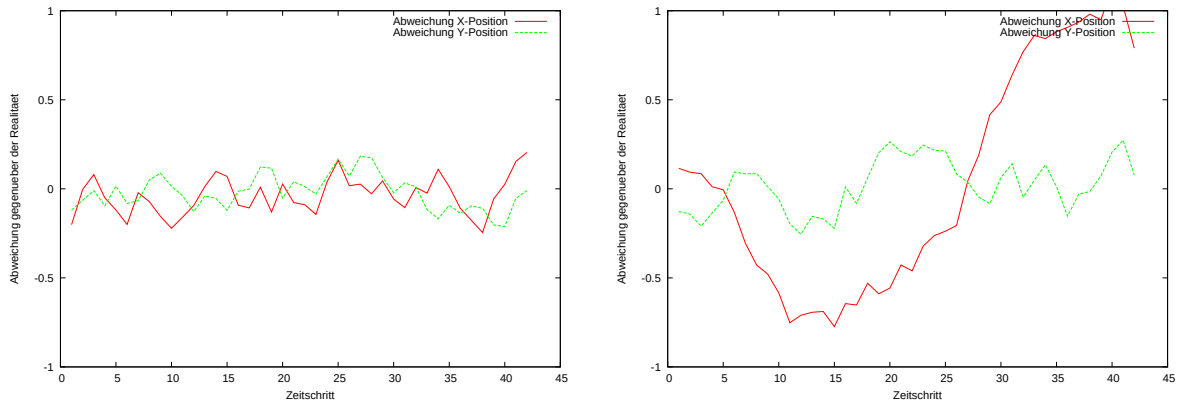


Abbildung 6.3: Abweichung der Position gegenüber der Realität mit linearem (links) und sinusförmigem Bewegungsmodell (rechts) bei einem Meter Rauschen

In Abbildung 6.3 sind die Abweichungen der getrackten von der realen Position zu sehen. Hierbei ist ein deutlicher Unterschied zwischen dem linearen und sinusförmigen Bewegungsmodell zu erkennen. Während die Abweichung des sich linear bewegenden Objekts um die X-Achse schwingt, ist bei dem sich sinusförmig bewegenden Objekt deutlich die Sinusschwingung zu erkennen. Dies leitet sich daraus her, dass das Bewegungsmodell des Filters eine Sinusschwingung nicht abbilden kann. Auch bei Betrachtung des Signal-Rausch-Abstandes wird dies deutlich. Der durchschnittliche Signal-Rausch-Abstand beträgt bei linearer Bewegung $\frac{S}{N} = 830.857$ für die X-Koordinate gegenüber $\frac{S}{N} = 166.887$ bei sinusförmiger Bewegung.

In Y-Richtung fällt der Unterschied mit $\frac{S}{N} = 422.462$ (sinusförmig) und $\frac{S}{N} = 643.709$ (linear) weniger gravierend aus. Dies ist dadurch zu erklären, dass die Sinusschwingung nur in X-Richtung schwingt, das Objekt sich aber konstant mit $10\frac{m}{s}$ in Y-Richtung bewegt. Dass sich das Verhältnis dennoch verschlechtert, liegt an der Koppelung der Größen durch den Kurswinkel und die Geschwindigkeit.

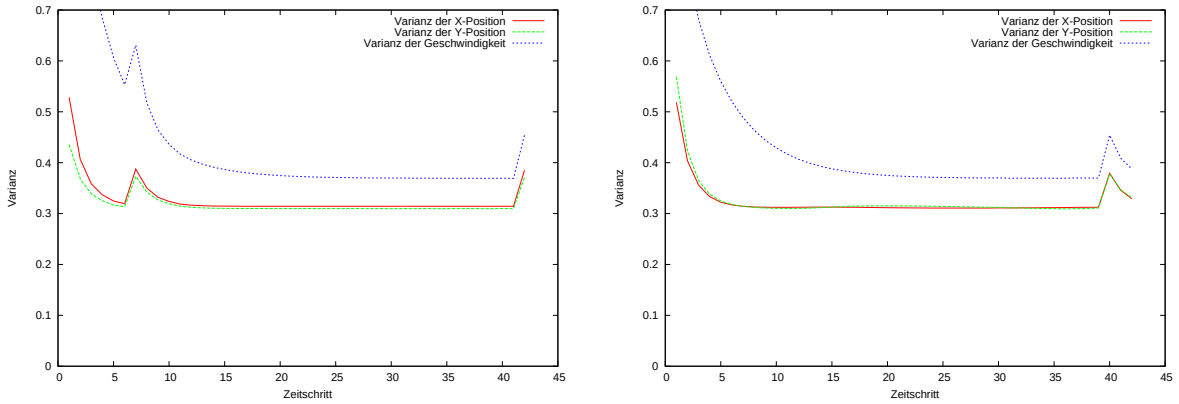


Abbildung 6.4: Varianzen der Zustandsgrößen mit linearem (links) und sinusförmigem Bewegungsmodell (rechts) bei einem Meter Rauschen

Auf die Varianzen haben die unterschiedlichen Bewegungen der Objekte keinen Einfluss. Dies ist in der Abbildung 6.4 zu erkennen. Hier sind die Varianzen der einzelnen Zustandsgrößen bei einem Rauschen von einem Meter aufgetragen.

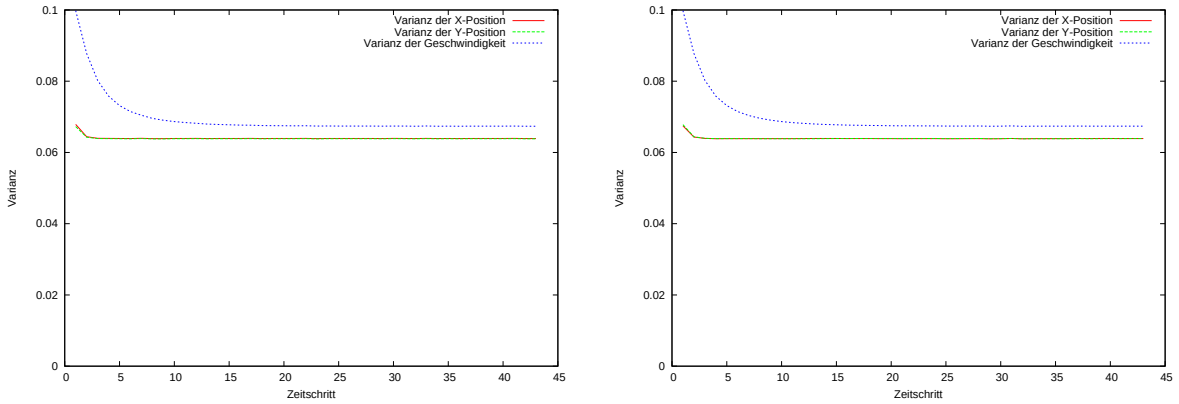


Abbildung 6.5: Varianzen der Zustandsgrößen mit linearem (links) und sinusförmigem Bewegungsmodell (rechts) ohne Rauschen

In Abbildung 6.5 sind ebenfalls die Varianzen der Zustandsgrößen dargestellt, diesmal jedoch ohne Rauschen. Hier ist deutlich zu erkennen, dass diese einen Wert von nahe Null annehmen.

6.4 Sichtbereiche der Sensorsysteme

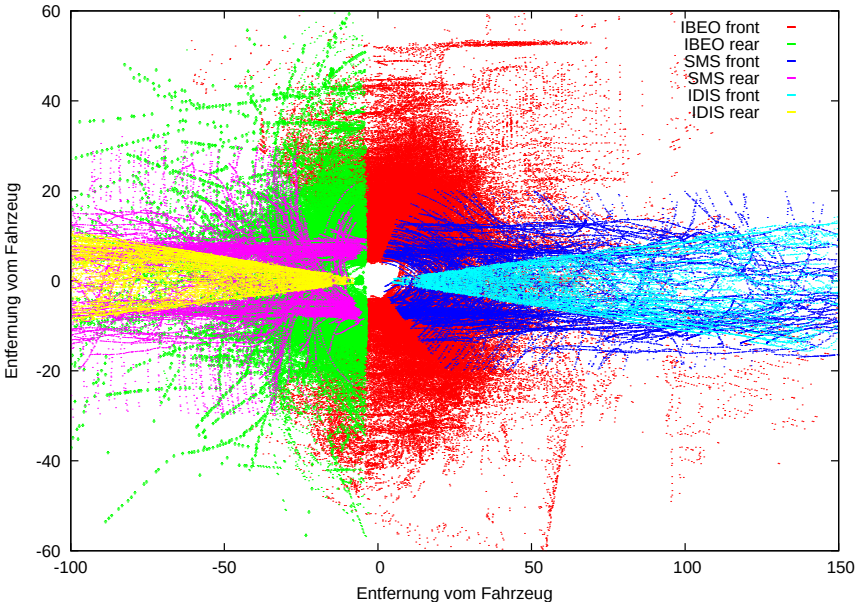


Abbildung 6.6: Sichtbereiche der Sensoren am Versuchsträger

In diesem Abschnitt wird die einzige Messung vorgestellt, die sich auf reale Messdaten stützt. Im Folgenden werden die Sichtbereiche der Sensoren visualisiert. Hierzu werden die durch die Sensoren erzeugten Konturpunkt im lokalen mitbewegten Koordinatensystem aufgetragen. Der Sichtbereich ist durch das Sensors-Modul auf 100m hinter dem und 150m vor dem Versuchsträger beschränkt.

Abbildung 6.6 trägt alle Konturpunkte, die während einer Messfahrt auf dem Versuchsgelände erzeugt worden sind, auf. Dabei sind in Rot und Grün die Bereiche der IBEO Sensoren zu erkennen. Hier ist ein Sichtbereich von ca. 40m zu erkennen, in dem gehäuft Objekte erkannt wurden. In Blau und Violett sind die Sichbereiche der SMS Radarsensoren dargestellt. Hier ist anfangs der früher beschriebene Öffnungswinkel zu erkennen. Da die Testfahrt in urbanen Gebiet stattfand, ist ein Korridor von ca. 40m zu erkennen, der den Abstand zwischen zwei Gebäuden darstellt. So konnte eine weitere Spreizung des Sitzbereichs nicht evaluiert werden. Die gelben- und cyanfarbenen Punkte bilden die Sichtbereiche der Infrarotsensoren ab. Auch hier ist der Öffnungswinkel deutlich zu erkennen. Sowohl bei den Radar- als auch bei den Infrarotsensoren kann die Sichtweite durch die oben beschriebene Einschränkung des Sensors-Moduls nicht bestimmt werden.

7 Ausblick

Die in dieser Arbeit beschriebene Software zur Multi-Sensor-Daten-Fusion hat während des DARPA Urban Challenge und diverser Testfahrten wiederholt bewiesen, dass sie das Rauschen der Sensoren sowie das Tracken einer Vielzahl von Objekten sehr gut beherrscht. Sie wurde während der Entwicklung der Fahrzeugfunktionen sowie während des Site-Visits (Viertelfinale) des Wettbewerbs erfolgreich eingesetzt.

Die Software läßt sich in unterschiedliche Richtungen erweitern. Zunächst wäre eine Erweiterung des PreTracking um den gleichen Ansatz, der im Haupttracking eingesetzt wird, wünschenswert. Hier könnte durch eine nicht starre Zuordnung eines PreTracks zu einer SensorID die Reaktionszeit verbessert werden.
Ein Ansatz zur Verbesserung des Haupttracking wäre der Einsatz von Multimodellkalmanfiltern. Diese bieten sich gerade für den Einsatz im urbanen Gebiet an, da die auftretenden Objekte sehr unterschiedlichen Bewegungsmodellen folgen.
Durch die Betrachtung des Umfelds in zwei Dimensionen lassen sich Sensorgeister, die durch Unebenheiten im Boden entstehen, nicht erkennen. Eine Betrachung des Höhenprofils der Umgebung könnte diese Geister eliminieren.

Eine generatorbasierte Erstellung einer Datenakquisesoftware der Sensoren und deren Integration in die Fusion kann eine weitere interessante Aufgabe darstellen. Auf diese Weise könnte der Einsatz von unterschiedlichen Sensoren anderer Hersteller oder der Einsatz der Software in unterschiedlichen Fahrzeugen einfacher realisiert werden.

Außerdem könnte die Portierung der Sensor-Daten-Fusion in Hardware eine lohnende Aufgabe sein. Diese müsste zum Ziel haben, unterschiedliche fahrzeugeigene Sensoren auf eine gemeinsame, für Fahrerassistenzsysteme nutzbare, Schnittstelle abzubilden. Hierzu könnte die Software zur Sensor-Daten-Fusion und Datenakquise beispielsweise in einem FPGA realisiert werden. Auf diese Weise wäre der Einsatz von neuartigen Fahrerassistenzsystemen in unterschiedlichen Fahrzeugen ohne große Umbauten am Fahrzeug möglich.

Eine alternative Herangehensweise an die Aufgabe der Umfelderkennung stellen Belegungsgitter (Occupancygrids) dar. Diese beschreiben die Umgebung nicht als Menge von Objekten,

sondern als 2D-Höhenfeld, das eine Wahrscheinlichkeit beschreibt, dass eine Zelle durch ein Hindernis belegt ist. Mit Hilfe dieses Ansatzes lassen sich Rohdaten von Sensoren, die in dieser Arbeit nicht verwendet wurden, besser verarbeiten. Ferner lassen sich nicht nur Aussagen über Hindernisse, sondern auch Aussagen über die Nicht-Existenz von Hindernissen treffen.

Die genannten Weiterentwicklungsmöglichkeiten sind beispielhaft und nicht vollständig. Es besteht noch ein großes Forschungs- und Entwicklungspotenzial im Umfelderkennungsbereich, das es in der Zukunft weiter zu bearbeiten gilt.

A Datenformate

A.1 LaneNetworkObject

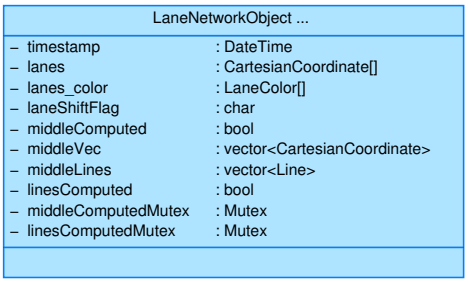


Abbildung A.1: Klassendiagramm der LaneNetworkObject Klasse

Die LaneNetworkObject Klasse enthält, wie in Abbildung A.1 beschrieben, eine Reihe von Attributen, die Polygone beschreiben, die die Fahrspur darstellen. Im Folgenden werden die Attribute beschrieben.

Attribut	Beschreibung
timestamp	Zeitpunkt, zu dem das Objekt instanziert wurde.
lanes	Dieser Array von vector<CartesianCoordinate> enthält eine Menge von Punktmengen, die eine Fahrspurlinie beschreiben. Der Array kann bis zu vier Punktmengen aufnehmen. Diese beschreiben von rechts nach links (vom Fahrzeug aus gesehen) jeweils eine Linie. Auf diese Weise ist es möglich, auch Informationen über Nebenspuren zu übermitteln.
lanes color	Dieses Attribut stellt einen weiteren Array dar. Er gibt entsprechend dem lanes Attribut die Farbe der Fahrspur (Gelb/Weiß) an.

Tabelle A.1: Attribute der Klasse LaneNetworkObject

Attribut	Beschreibung
laneShiftFlag	Das LaneShiftFlag gibt, an ob zwischen dem aktuellen und dem letzten LaneNetworkObject ein Fahrspurwechsel erkannt wurde. Die Werte können dabei ± 1 betragen.
middleVec	Punkte der Fahrspurmitte, die aus bekannten Daten berechnet werden.
middleComputed	Wahr, wenn die Daten in middleVec aktuell sind.
middleComputedMutex	Mutex, um die Punktmenge zu schützen.
middleLines	Menge von Lines zur Berechnung der Fahrspurmitte.
linesComputed	Wahr, wenn die Daten in middleLines aktuell sind.
linesComputedMutex	Mutex, um die Menge von Linien zur Berechnung der Fahrspurmitte zu schützen.

Tabelle A.1: Attribute der Klasse LaneNetworkObject (Fortsetzung)

A.2 SensorSweep und SensorObject

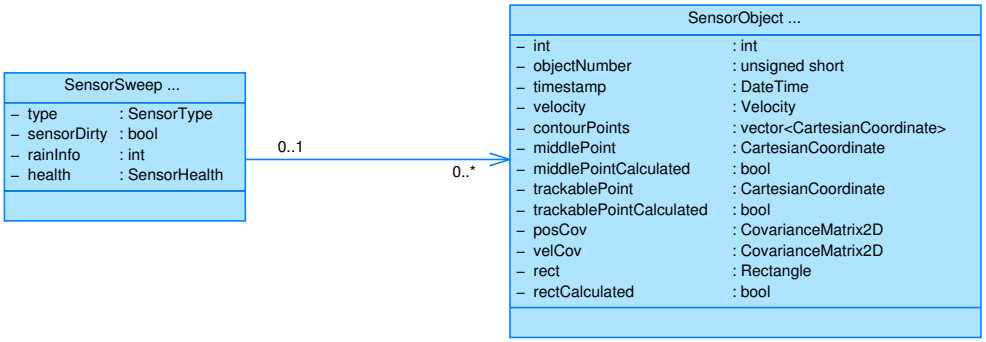


Abbildung A.2: Klassendiagramm der SensorSweep und SensorObject Klassen

Die Klassen SensorSweep und SensorObject (siehe Abbildung A.2) werden genutzt, um Sensordaten aus dem Sensors Modul zu dem Fusions Modul zu übertragen. Sie stellen die gemeinsame Datenstruktur dar, in der alle Sensordaten konvertiert werden müssen, um von der Sensor-Daten-Fusion verarbeitet zu werden. Zu diesem Zweck wird ein SensorSweep pro Sensordurchlauf erzeugt. Ihm werden alle dabei anfallenden Sensorobjekte in Form von SensorObjects hinzugefügt. Im Folgenden werden die Attribute dieser Klassen näher erläutert.

Attribut	Beschreibung
type	Der Type gibt den Sensor an, von dem die Daten erzeugt wurden. Auf Grund dieser Information werden später beispielsweise Kalmanfilter ausgewählt oder ID Domänen bestimmt. Er kann Werte eines Enum enthalten, der alle Sensorarten abdeckt.
sensorDirty	Dieses Flag gibt an, ob der Sensor verschmutzt ist und gereinigt werden sollte.
raininfo	Das Raininfo Attribut gibt die durch die IBEO Laserscanner errechneten Regenwerte weiter. Die Werte nehmen hier den Bereich von 0 bis 255 ein. 0 bedeutet „kein Regen", 255 „nicht verfügbar" und 1 bis 254 „Regenpunkt pro 1000".
health	Dieses Attribut übermittelt Informationen über den Zustand des Sensors. So lassen sich beispielsweise die Taktzeiten oder Informationen über den Betriebszustand des Sensors ermitteln.

Tabelle A.2: Attribute der Klasse SensorSweep (Fortsetzung)

Attribut	Beschreibung
id	Die ID ist der Identifikator für ein Sensorobjekt. Er wird von dem jeweiligen Sensor vergeben und ist für einen Sensortyp eindeutig. Er kann Werte zwischen 1 und 999 annehmen.
objectNumber	Dieses Attribut ordnet ein Sensorobjekt in einer Menge von anderen Sensorobjekten eines SensorSweeps.
timestamp	Auftrittszeitpunkt des Objekts.
velocity	Geschwindigkeitsvektor des Objekts.
contourPoints	Dieser vector<CartesianCoordinate> gibt die Kontur des Objekts an. Er bestimmt auch maßgeblich die berechneten Werte, wie Hüllquader oder Mittelpunkt.
middlePoint	Der Mittelpunkt des Hüllquader des Objekts.
middlePointCalculated	Wahr, wenn der Wert in middlePoint aktuell ist.

Tabelle A.3: Attribute der Klasse SensorObject

Attribut	Beschreibung
trackablePoint	Der trackablePoint ist ein verhältnismäßig stabiler Punkt auf der Kontur des Objekts. Die genaue Berechnung wird in Listing 4.3 dargestellt. Er wird im Pre-Tracking Modul genutzt, um ein Objekt zu verfolgen.
trackablePointCalculated	Wahr, wenn der Wert in trackablePoint akuell ist.
posCov	Dieses Attribut gibt die Kovarianz der Positionsdaten des Objekts an.
velCov	Das velCov Attribut gibt die Kovarianz der Geschwindigkeit an.
rect	Hüllquader des Objekts.
rectCalculated	Wahr, wenn rect aktuelle Daten enthält.

Tabelle A.3: Attribute der Klasse SensorObject (Fortsetzung)

A.3 FusionObject

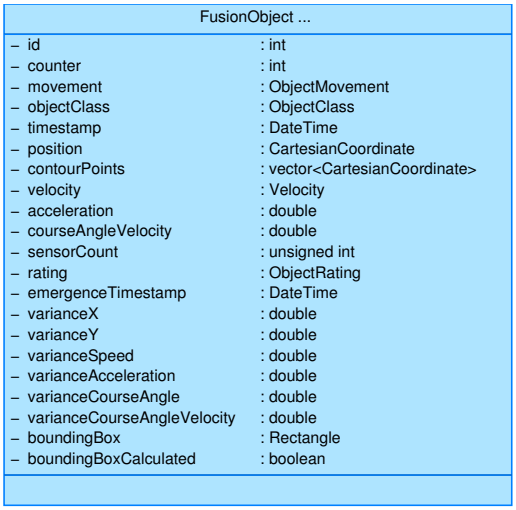


Abbildung A.3: Klassendiagramm der FusionObject Klasse

Die FusionObject Klasse (siehe Abbildung A.3) stellt die Informationen dar, die von der Sensor-Daten-Fusion an die digitale Karte übermittelt werden. In ihr sind Informationen über Art und Ausprägung eines Objekts beschrieben. Im Folgenden wird auf die Attribute eingegangen.

Attribut	Beschreibung
id	Identifikator des Tracks, von dem das FusionObject seine Informationen erhält.
counter	Anzahl der Bestätigungen des Sensortracks.
movement	Das movement Attribut enthält Informationen über die Art der Bewegung des Objekts. Es kann die Zustände unknown, static und dynamic annehmen. Der exakte Algorithmus zur Bestimmung dieses Wertes ist in Abschnitt 4.3.5 angegeben.
objectClass	In diesem Attribut wird die Art des Objekts angegeben. Es kann die Werte unknownSmall, pedestrian, car, truck, unknownBig und notClassified annehmen. Dieses Attribut wird im Rahmen dieser Arbeit nicht genutzt.
timestamp	Zeitpunkt der letzten Daten, die in diesen Track eingeflossen sind.
contourPoints	Konturpunkte des Tracks.
velocity	Der Geschwindigkeitsvektor in $\frac{m}{s}$ des Tracks.
acceleration	Hier wird die Beschleunigung des Objekts in $\frac{m}{s^2}$ abgebildet.
courseAngleVelocity	Mit diesem Attribut wird die Kurswinkelgeschwindigkeit in Rad übermittelt.
sensorcount	Das sensorcount Attribut gibt die Anzahl der Sensoren an, die der Erstellung dieses Tracks zugrunde liegen.
rating	Das rating stellt eine Aussage über die Qualität eines Tracks dar. Der genaue Algorithmus wird in Abschnitt 4.3.5 beschrieben.
varianceX	Varianz der X-Position.
varianceY	Varianz der Y-Position.
varianceSpeed	Varianz der Geschwindigkeit.
varianceAcceleration	Varianz der Beschleunigung.
varianceCourseAngle	Varianz des Kurswinkels.
varianceCourseAngleVelocity	Varianz der Kurswinkelgeschwindigkeit.
boundingBox	Hüllquaders des Tracks.
boundingBoxCalculated	Wahr, wenn boundingBox aktuelle Daten enthält.

Tabelle A.4: Attribute der Klasse FusionObject (Fortsetzung)

Literatur

[18600] Federal Information Processing Standards Publication 186-2. Digital signature standard (dss). Technical report, U.S. Department of Commerce/National Institute of Standards and Technology, January 2000. FIPS PUB 186-2.

[Age06] Defense Advanced Research Projects Agency. Darpa announces third grand challenge: Urban challenge moves to the city, May 2006.

[Bal99] Dirk Balzer. *Online-Demodulation start gestörter winkelmodulierter Signale mit dem Extendet Kalman-Filter*. PhD thesis, Universität-Gesamthochschule Siegen, Fachbereich Elektrotechnik und Informatik, May 1999.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented software architecture - A system of patterns*. Wiley, 1996.

[Bro98] Eli Brookner. *Tracking and Kalman Filtering Made Easy*. John Wilkey & Sons, Inc, 1998.

[DNS+06] Stéphane Ducasse, Oscar Nierstrasz, Nathanael Schärli, Roel Wuyts, and Andrew Black. Traits: A mechanism for fine-grained reuse. In *ACM Transactions on Programming Languages and Systems*, volume 28, pages 331–388, March 2006.

[ebn96] Iso/iec 14977 information technology - syntactic metalanguage - extended bnf. Technical report, ISO, December 1996.

[Eff02] Jan Effertz. *Multi Sensor Joined Probabilistic Data Association zur Objekterkennung und Datenfusion in einem Multi-Sensor-Netzwerk*. Institut für Regelungstechnik Technische Universität Braunschweig, October 2002.

[FWM94] Geoffrey C. Fox, Roy D. Willams, and Paul C. Messina. *Parallel Computing Works*. Morgan Kaufmann, 1994.

[Hel06] Hella KGaA Hueck & Co, Rixbecker Str. 75, 59552 Lippstadt. *Infrared Sensor for Lidar Based Distance Measurement*, May 2006.

[HMU03] John e. Hopcroft, Rajeev Motwani, and Jeffery D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson Education International, second edition, 2003.

[Hu62] Ming-Kuei Hu. Visual pattern recognition by moment invariants. In *Information Theory, IEEE Transactions on*, volume 8, pages 179– 187. IEEE, Feburary 1962.

[Ibe06] Ibeo Automobile Sensor GmbH, Fahrenkrön 125, 22179 Hamburg. *ALASCA User Manual*, March 2006.

[Inc06] Silicon Graphics Inc. Standard template library programmer's guide, 2006.

[Kal60] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.

[oD04] Department of Defense. World geodetic system 1984. Technical Report Third Edition, Nationl Imaging and Mapping Agency, Januar 2004.

[ope07] *OpenCV*, 2007. [Online; accessed 14-May-2007].

[pos04] *The Open Group Base Specifications Issue 6 IEEE Std 1003.1.* IEEE, 2004.

[psq06] *PostgreSQL 8.2.0 Documentation*. The PostgreSQL Global Development Group, 2006.

[sma06] smart microwave sensors GmbH, Mittelweg 7, 38106 Braunschweig, Germany. *Universal Medium Range Radar Documentation*, October 2006.

[Ste05] Ryan Stephens. What is an iterator in c++ ? *O'Reilly Network*, October 2005.

[Wik07a] Wikipedia. Create, read, update and delete — wikipedia, the free encyclopedia, 2007. [Online; accessed 14-April-2007].

[Wik07b] Wikipedia. Exponential backoff — wikipedia, the free encyclopedia, 2007. [Online; accessed 13-May-2007].